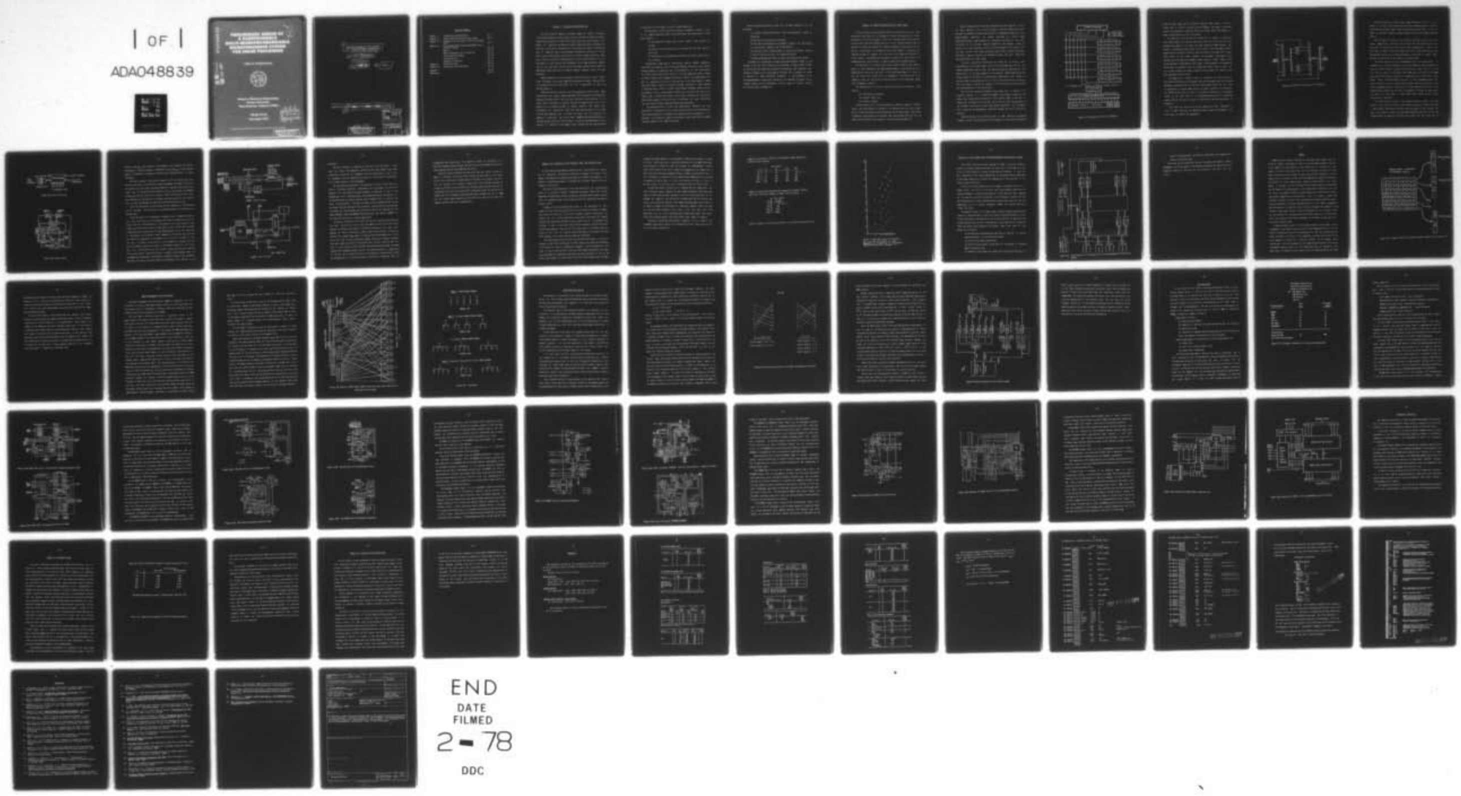
AD-A048 839 PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGI--ETC F/G 9/4
PRELIMINARY DESIGN OF A PARTITIONABLE MULTI-MICROPROGRAMMABLE M--ETC(U)
NOV 77 J F BOGDANOWICZ AFOSR-74-2661
UNCLASSIFIED TR-EE77-42 NL

| OF |

ADA048839



END
DATE
FILMED
2-78
DDC

0
NU

PRELIMINARY DESIGN OF A PARTITIONABLE MULTI-MICROPROGRAMMABLE MICROPROCESSOR SYSTEM FOR IMAGE PROCESSING

Julius F. Bogdanowicz

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE 77-42

November 1977

Supported by the AFOSR Grant 74-2661 and the NSF Grant ENG 76-18567.

6

Preliminary Design of a Partitionable Multi-Microprogrammable Microprocessor System for Image Processing,

by

10 Julius F./Bogdanowicz

14 TR-EE 77-42

11 Nov[redacted] [redacted]77

12 72 p.

Supported by the AFOSR [redacted]-74-2661, [redacted] NSF [redacted]-ENG-76-18567

15

292 000

Table of Contents

Chapter I. Introduction and Motivation

The use of digital computers to process images has rapidly increased. It has been estimated [1] that within the next ten years a potential digital image processing market of 400 million frames per year could develop. This market will depend on how powerful digital image processing techniques become and if they can be implemented in a cost-effective manner. Although the cost of computing power has decreased, disregarding inflation, it is felt that a specialized computer architecture tailored to image processing might give at least two orders of magnitude gain in cost effectiveness. As will be discussed in a later section, the throughput for an image processor, depending on the algorithm and the size of the image, can be several orders of magnitude greater than that of a general purpose computer doing the same operation.

Image processing in its most general sense consists of a set of operations performed on an input image which produces an output. However,there is no consensus of what constitutes the set of operations or what the desired output is.

Image processing, in general, can be separated into three areas, image digitization and coding, image enhancement and restoration, and image segmentation and description. The first area consists of the conversion of an image from a continuous to discrete form and the compression of this information so as to conserve storage or channel capacity. The second area, enhancement and restoration, deals with improving the quality of an image that has been degraded due to noise, blurring, lack of constrast, or geometric distortions. The third area, segmentation and description, is perhaps the most difficult of the three. It involves the measurement of properties, or features, of the image or parts thereof and the classification or description of the image in terms of these properties.

The motivation of this research is based on a number of factors. First of all, image processing differs from conventional data processing in three major aspects:

(1) two-dimensional image data arrays require large amounts of storage;

(2) high on-line processing rates are needed for real-time applications; and

(3) operations to be performed on the data are usually highly parallel in nature.

Processing of image data by conventional general purpose computers, those which are based on the so called Von Neumann architecture [2], require enormous amounts of computing time. The unsuitability of the architecture for doing these tasks, which results in the high computing costs, is based on several factors [3]. Programs and data are stored in the same memory unit and all operations are serially executed. Due to the large amount of image data, main memory capacity is usually exceeded, resulting in large amounts of overhead time in order to transfer data between secondary storage and the main memory. Finally, input/output transfers are slow since the central processor must initiate the transfer of each word. Therefore, by developing a specialized computer architecture that is optimized for image processing, real-time cost-effective image processing may be possible.

The second motivation is due to the technological advances that have been made in integrated circuits, along with the "birth" of microprocessors. Since microprocessors are becoming less expensive and more powerful, is it possible to use them as a means of providing a large, cost-effective computational capability for image processing.

With the above motivations in mind, the following questions will be discussed.

(1) In what configuration should a multi-microprocessor system be organized?

(2) How should the memory be allocated?

(3) How many microprocessors are needed to surpass the performance of a conventional computer like the PDP-11/70 ?

(4) How would such a system compete in terms of performance against a cellular logic array built for image processing?

(5) What advantages are there in using a multiprocessing system ?

In Chapter II, an approach to image processing hardware based on the concept of cellular logic arrays will be evaluated. A study by Duff, Cordella, and Leviadi [2] comparing the parallel processing and sequential processing of images will be investigated in Chapter III. In Chapter IV, another approach to image processing hardware, a partitionable multi-microprogrammable microprocessor system will be examined. In Chapter V, performance issues will be investigated. Finally, areas for further research will be discussed in Chapter VI.

Chapter II. Image Processing Cellular Logic Arrays

A cellular logic array processor consists of an array of cells or subprocessors, where each cell is assigned to a pixel (The word pixel is an abbreviation for "picture element".). Each cell receives inputs from an external data source and from neighboring cells. A cell contains boolean logic to process the inputs and storage for saving intermediate results prior to producing a final output. The storage for grey level images is in terms of encoded bit planes. For example, an 8 level grey scale image can be stored in 3 bit planes. Two outputs are available from each cell. One connects with neighboring cells and the other is used to output the processed pattern.

The structure of a cellular logic array is best suited for the performance of "local operations" on image neighborhoods. A local operation on an image defines a value for each pixel in the transformed image in terms of its own value and a small set of its neighbors. Since each pixel is assigned to a cell, all the values of the new pixels in the transformed image can be computed in parallel.

The optimization of a cellular logic array structure is based on three factors:

(1) interconnection pattern,

(2) internal logic, and

(3) internal storage.

An array which has a rich interconnection pattern, powerful internal logic, and large amounts of storage will be expensive but will have a capability of fast and sophisticated processing. On the other hand, if the array is sparsely interconnected with minimal logic and storage, each cell will be simple and inexpensive with, however, little processing capability.

Several image processes have been designed which are based on a cellular logic approach. The Illiac III [5,6] is a digital computer which was designed for automatic scanning and analysis of homogeneous image data, e.g. bubble-chamber negatives. The system consisted of 3 parts: image acquistion/display, image encoding for information transmission, and classification of the encoded image. A schematic of the computer is shown in Figure (1). It was recognized in the initial development of the system that the data rate for local image processing would create a system bottleneck. Therefore, a special processor was designed to permit the rapid recognition and description of an input image.

This unit, the Pattern Articulation Unit (PAU) was to perform local preprocessing on the input image, such as track thinning, gap filling, line element recognition, and so forth. The logical design was optimized for the idealization of the input image to a line drawing. Nodes representing end points, points of inflection or intersection, among others, were labled in parallel under the program control of a unit called the Taxicrinic Processor (TP). The output of the PAU is a graph, in a list structure, which describes the interconnection of the labled nodes.

The Taxicrinic Processor assembles these graphs into a coherent list structure subject to a recognition grammer. This grammer categorizes the graph, thus, recognizing the contents of the original image. In addition to controlling the operation of the PAU, the TP also oversees the operation of the arithmetic units and initiates input/output operations. The Arithmetic Unit is used for any mathematical operations needed, such as for statistical analysis.

The PAU consists of an iterative array of 1024 identical processing elements called stalactites which are arranged in a 32 by 32 two dimension

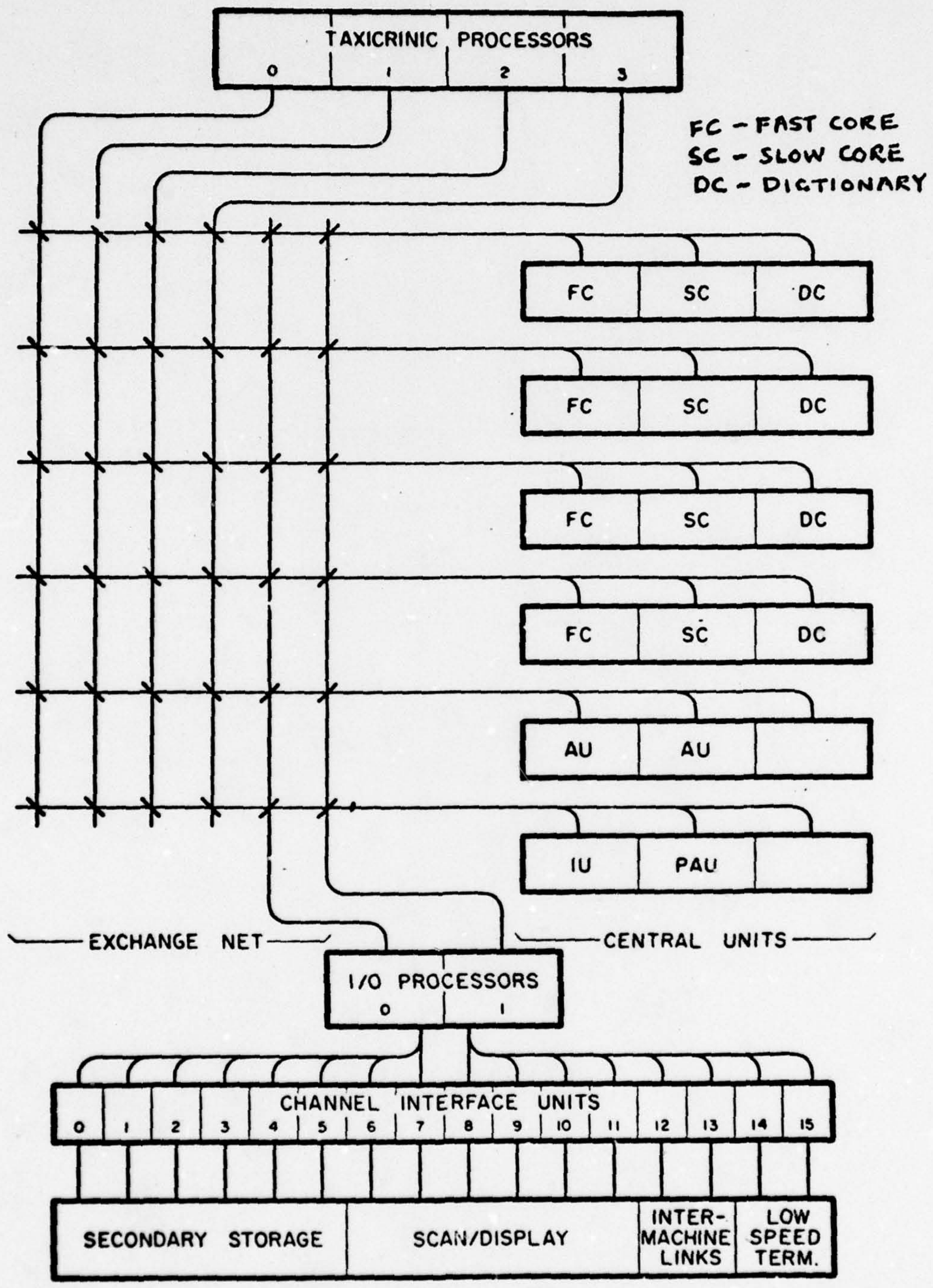


Figure (1) Structure of Illiac III Computer array. An input image, which is typically 1024 by 1024 pixels, is partitioned into a lattice of 32 by 32 pixel windows. The image is processed *serially by window and in parallel* within the window. Within the window, an image is described by a set of bit planes.

In Figure (2), the structure of a stalactite is shown in simplified form. Each element can accept an input from itself and from any of the eight neighboring elements in the plane. The input signals are ORed together, optionally complemented, and stored in one or more of the nine memory planes. Communications with extra planes in the core buffer is through the M plane, which serves as the buffer register of this memory. The outputs of any selected set of planes can be ANDed or ORed to get an output signal which can be optionally complemented. This signal is then passed on to neighboring stalactites. In addition, there is a signal path which allows an input signal to pass through the element without iterim storage. This feature allows path building with the array.

The common set of control lines is connected to each stalactite. Thirty-one instructions are provided. They can be grouped into the following categories concerned with (1) loading an input image string into the array or generating an output image string, (2) planar transfers of information, (3) redefining the value of a pixel in a plane on the basis of its neighbors either in the same plane or in corresponding positions for neighboring planes, and (4) the input or output of associatively derived coordinate information.

On a historical note, the Illiac III computer was never completed. A fire in 1967 destroyed the PAU and the main frame of the computer. At a later time, the project was abandoned.

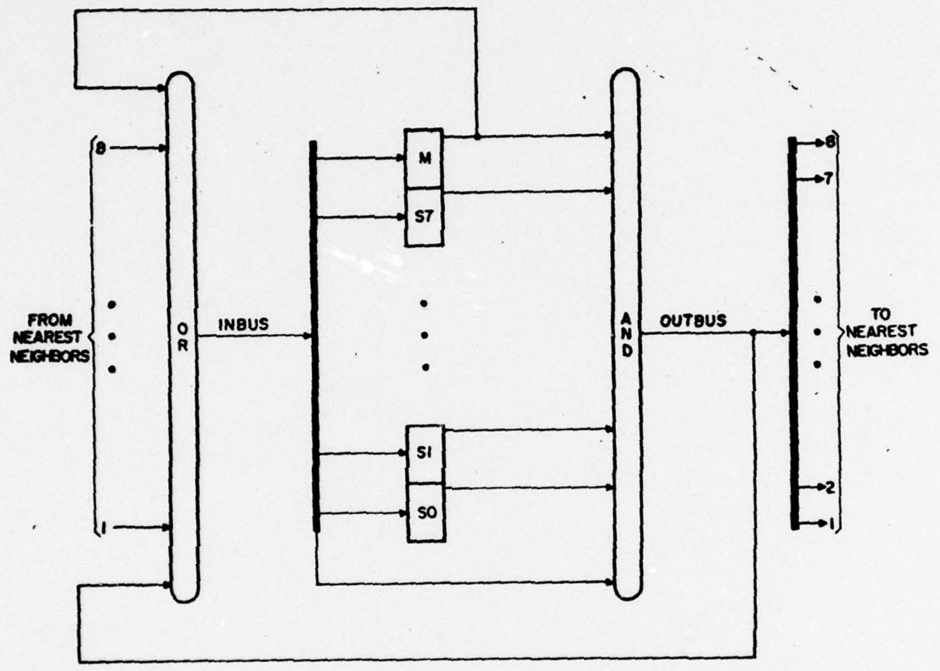


Figure (2) Simplified structure of a Stalactite

CLIP, an acronym for Cellular Logic Image Processor, is the name of a family of cellular logic arrays that have been proposed and constructed at the University College London. These processors have been built in an attempt to provide a general purpose hardware facility for image processing studies.

The CLIP family consists of four different processors, each with different capabilities. The CLIP 1 [7] served to test the feasibility of constructing an integrated circuit array in which propagation of signals into and from all parts of the array might take place. Three functions could be implemented: extraction of connected objects, extraction of object boundaries, and extraction of the contents of closed loops.

The CLIP 2 [8] was the first programmable array to be constructed. It consisted of a 16 by 12 hexagonal array of 192 symmetric boolean operators. Each cell had two inputs: A_0 being the value of the input pattern at the cell and A_n which is the output from a NAND gate. The inputs to the NAND gate are the A^1_1 outputs from the six neighboring cells in the array. These binary inputs are transformed by the cell into two independent binary outputs, A_1 and A^1_1. The A_1 is the processed pattern and the A^1_1 connects with the six neighboring cells. Figure (3a) is a logic diagram of the cell. Since communications between the cell and its neighbors is through a NAND gate, the source of the signal is never identifiable. Thus, the cell is nondirectional in character.

In Figure (3b), the layout of the complete system is shown. An input pattern A_o is stored in a 192 bit shift register M_{in} as a sequence of 0 and 1 logic states, representing the white and black parts of the original image. This pattern can be circulated and displayed on an oscilloscope. The output pattern A_1 appears on the 192 output leads from the array and is

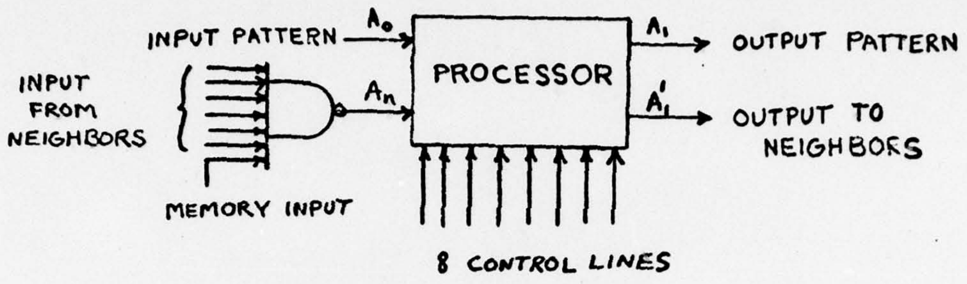


Figure (3a) CLIP 2 Cell Logic

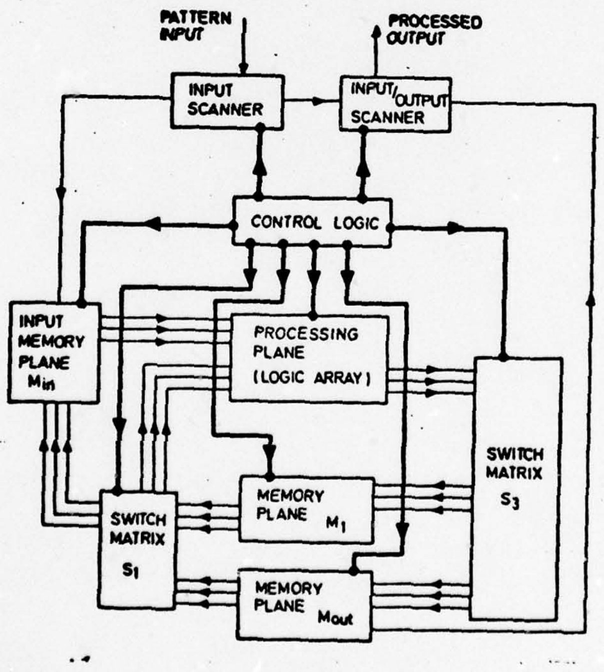


Figure (3b) System Layout

stored in the M_{out} shift register. The contents of this register can also be displayed on the oscilloscope. In addition to M_{in} and M_{out}, a third 192 bit memory is provided for storage of a pattern while another pattern is being processed.

The CLIP 2 system is controlled by means of a 12 bit instruction word. There are two types of instructions, LOAD and PROCESS. The LOAD instructions are used to cycle the input and output memories, load and display their contents, and for transferring the contents of M_{in} or M_1 or both. The PROCESS instructions set the values of the control lines, the position of the routing switches S_1, the position of the gate switches S_2, and set up a logical 0 or 1 on the interconnection leads which extend outside the array.

Up to 32 instructions can be stored, so that sequences of instructions can be executed. The instructions are entered either by 12 switches or by punched tape.

Since the CLIP 2 cell represents a compromise over a completely general cell, the versatility of this processor was severly limited due to the nature of the cell interconnections. In the general cell, there would be nine inputs, two outputs and a total of 1024 control lines. In the CLIP 3 [9,10], a compromise was achieved which allowed implementation of all the functions of the general cell by means of short sequences of functions, but which did not require the complexity of the general cell. This cell, as shown in Figure (4a), is similar to the CLIP 2. It is preceded by a threshold unit fed by the 8 neighbor inputs; each input is individually gated into the threshold unit. The threshold unit has 3 control inputs which set the threshold level, and 8 control inputs which select a subset of the neighbor inputs to be summed and thresholded. The processor requires 8 control lines as before. Therefore, 19 control lines are required to determine the function to be

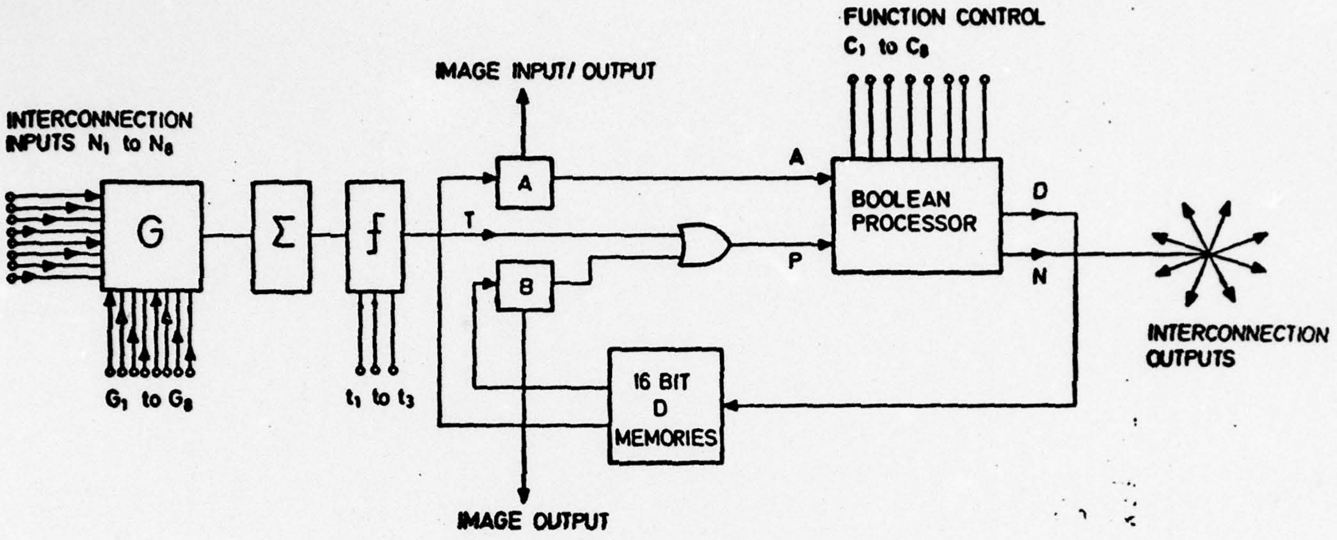


Fig.(4a) CLIP 3 cell logic

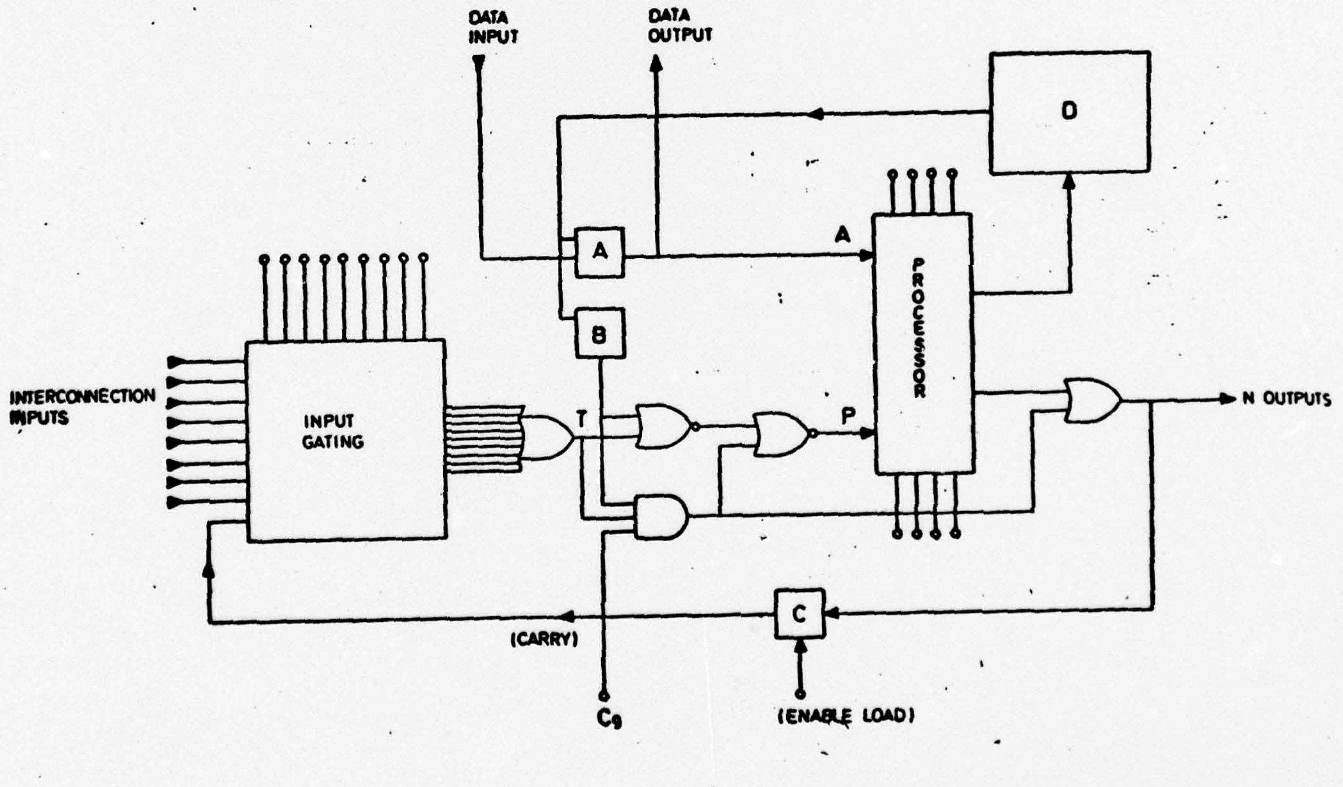


Fig.(4b) CLIP 4 cell logic

performed.

The array structure is comprised of 192 cells, as in the CLIP 2. However, the array interconnection pattern can be either square, with eight connections to each cell, or hexagonal, with six connections to each cell, or hexagonal, with six connections to each cell. The choice of interconnection is under control of the programmer.

The two patterns presented to the processor are held in the A and B registers. Processed patterns, represented by D, are output into one of 16 memories. The interconnection output N appears as inputs N_1 to N_8 in the neighboring cells' threshold gates. The thresholded sum of the inputs is T. The OR gate forms the logical sum B+T, which together with A, provide the input to the processor. Patterns are input into the system and the results are output from the system via the A register. Instructions to the processor are stored in a 256 word RAM. The instructions are 24 bits long and are of 5 types: PROCESS, LOAD, and BRANCH instructions and two special BRANCH instructions for entering or leaving subroutines.

An obvious limitation in the CLIP 3 is the low resolution obtained with the small 16 by 12 cell array. Also, the number of grey levels that can be handled is limited by the number of D-arrays available at any given time. Because of these limitations, a hybrid CLIP 3 array was built to gain experience with larger image areas. A scanning unit was designed which interfaces the CLIP 3 with a television camera. Provision is made to threshold the video signal and digitize a 96 by 96 array. The unit scans the 192 cell CLIP 3 array across the 96 by 96 data field and provides storage to handle signals that propagate between sectors. The scanning process is complicated by the fact that a forward scan must be followed by a reverse scan to allow for propagations in all directions and a check scan to determine that all propagations have taken place. The complete system is interfaced to a PDP-11/10 computer which provides data and instruction storage and also provides program editing and assembling facilities.

Using the results obtained from the CLIP 3 and the hybrid version, a NMOS LSI processor called the CLIP 4 [11,12] has been built. It is a 96 by 96 cell array with several changes made in the basic cell design. The structure of the basic cell is shown in Figure (4b). The D storage has been increased to 32 bits; the interconnection threshold gate has been replaced by an OR gate; and a few extra gates and an additional buffer to provide an automatic carry for arithmetic operations has been included. The use of NMOS LSI will cause a speed reduction in the CLIP 4 by a factor of at least five compared to the CLIP 3 which used MSI TTL.

Chapter III. Evaluation of the Cordella, Duff, and Levialdi Study

A study that was performed by Cordella, Duff, and Levialdi [4] comparing the sequential and parallel processing of images is discussed in this chapter. The sequential processing was done on an HP2116 minicomputer. The parallel processing was done on a hypothetical processor, based on the performance of the CLIP 3.

The tasks investigated were of the type used during the preprocessing phase of a pictorial pattern recognition procedure. These tasks include smoothing, thresholding, contour extraction, thinning, and perimeter evaluation. An assumption made in the study was that the images had a maximum of 32 grey levels.

A number of serious problems were found in the evaluation of their study. The first area concerns the use of clock cycles as a basis of comparison. Unless the time for a clock cycle is equal in both p processors, the results are not a valid indication of how long a task takes. For example, consider the case where Processor A has a clock cycle of 1 microsecond and Processor B has a clock cycle time of 10 microseconds. If both processors use the same number of clock cycles for a task, then it is obvious that Processor B will take 10 times as long as Processor A to complete the same task. However, their study would not show any difference.

The second problem area involves how realistic the study is in terms of present or near future technology. In their study, they used the performance of the CLIP 3 as a basis of comparison. However, it is not feasible at this time to build large cellular arrays with this speed performance. For example, using MSI TTL technology from which the CLIP 3 was built, a 96 by 96 array would require 16 racks, each 6 feet tall by 19 inches, plus the space required for power supplies. As discussed in the previous chapter, in order to build a 96 by 96 array it required switching to an LSI NMOS technology. This resulted in a factor of five loss in speed for LOAD/PROCESS instructions and a factor of ten loss in speed for propagation. This speed loss is due to the fact that gate propagation delay in NMOS is much greater than in TTL. Although it is possible to place approxiately 1000 gates on a LSI chip as compared to 100 for MSI, it is still only possible to place 8 cells on one chip. Thus, a 7 foot tall rack is required to hold the entire array.

As mentioned before, Cordella et al. assumed that the images were limited to 32 grey levels. However, for most image processing situations, grey levels of 128 or 256 are more realistic. In fact, it was found that by extending the equation they derived for thresholding [13] to arbitrary grey levels, for a 50 by 50, 256 grey level image, assuming equal cycle times, it is faster to compute the threshold on a sequential computer (Table (1)) than on a cellular logic processor. This is due to the fact that since CLIP operations are low level, overhead results when doing higher level operations. The overhead becomes significant when a small image is computed.

Therefore, considering the problems presented in this chapter, the tremendous speed gains (Figure (5)), determined in their study, must be taken in the proper perspective.

Number of Clock Cycles $(X10^3)$ for Thresholding (MXM) Image Using Cellular Logic Processor

	Grey Levels			
M	32	64	128	256
50	12	24	47	93 ←
100	24	46	92	183
500	113	226	450	899
1000	225	500	898	1800

Number of Clock Cycles Required for Sequential Processor (Values Vary Very little with Respect to Number of Grey Levels)

	Grey Levels
M	32-256
50	75 ←
100	300
500	7500
1000	30000

Table (I) Number of Clock Cycles Required as a Function of the Grey Scale

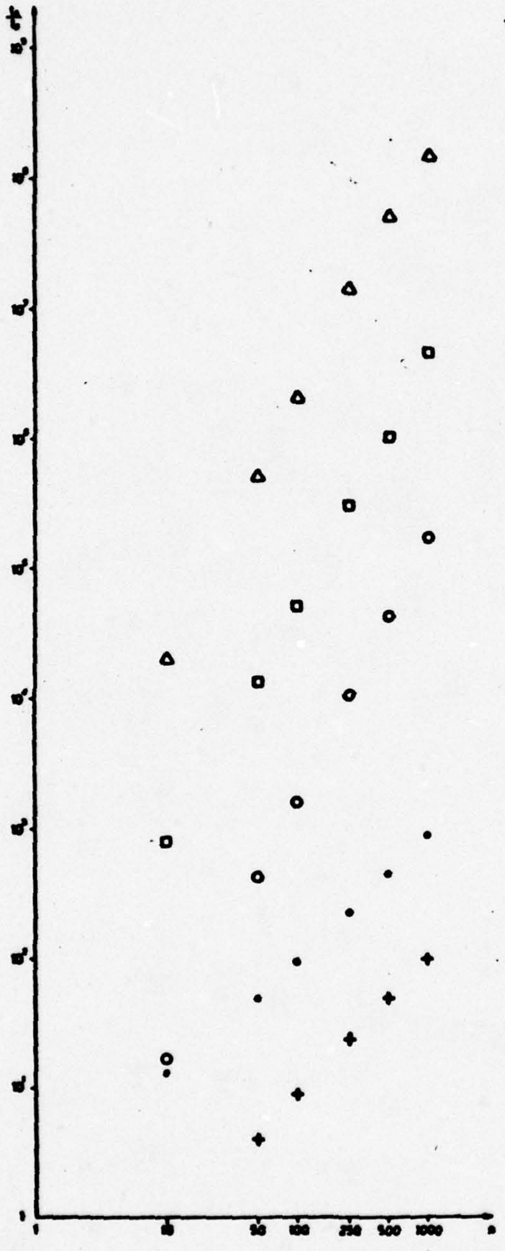


Figure 5- Speed gain (t_s/t_p) as a function of n for each of the following tasks: smoothing (o), thresholding (+), thinning(△), contour extraction (□), perimeter (•).

Chapter IV. Partitionable Multi-Microprogrammable Microprocessor System

This chapter discusses another approach to image processing hardware. Instead of having a boolean processor for each pixel in the image as in the Illiac or CLIP processors, a sophisticated powerful processor is used for each subsection of the image. Computations are performed on each subsection of the image sequentially, however the computation of the algorithm is effectively done in parallel.

If the size of each subsection of the image is allowed to shrink to a single pixel, the result is a cellular logic array computer which was discussed in Chapter 3. A specialized computer of this form has been proposed for the numerical solution of problems in fluid mechanics [14,15]. The computer described has a fixed interconnection network where each cell can only communicate with its nearest neighbors "above" and "below" and to the "right" and "left".

The approach taken in this paper seeks a balance between organizational complexity and performance. Since this system is being developed for a particular application, some generality can be sacrificed for an improved performance based on the specific requirements of the a application. Figure (6) shows the overall block diagram of the system. Some of the goals of this design are as follows:

(1) use multiple microprocessors operating in parallel to achieve high performance in a cost-effective manner,

(2) provide high speed input/output,

(3) have a flexible memory system which is structured to minimize memory contention, and

(4) design an interconnection network that allows partitioning of a

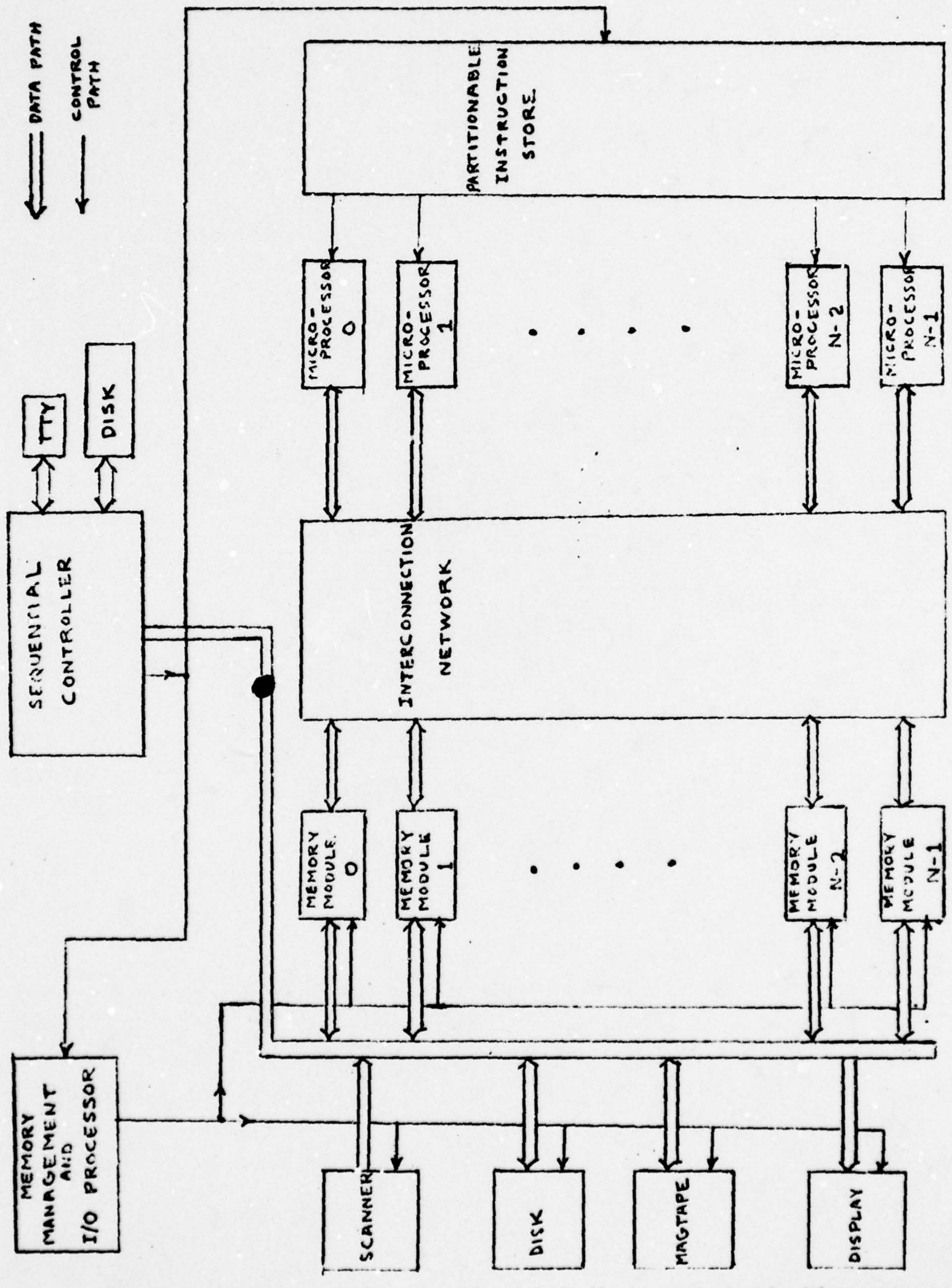


Figure (6) **Block diagram of Partionable Multi-microprogrammable Microprocessor System**

group of microprocessors into smaller groups where each subgroup can work on a different task.

The system can be broken down into the following parts: Memory, Memory Management and I/O Processor, Interconnection Network, Microprocessors, and Sequential Controller. Each part will be discussed in the next five sections.

Memory

Memory in this system is divided into two types, data memory and instruction memory. Data memory is split up into N separate modules, where N is the number of microprocessors in the system. Data is allocated to each module in terms of a row major form. This is illustrated in Figure (7). Each module contains sufficient memory to store the original subsection of the image as well as storage for the resulting computation, temporary variables, and a buffer to allow communication between microprocessors. The exact amount of storage will be a function of the largest image to be processed and the number of microprocessors in the system. By distributing in row major form, the number of different memory modules with which each microprocessor must communicate can be minimized. This, in effect, reduces the overhead necessary to control the interconnection network between microprocessors and memory. The address mapping function of the memory management and I/O processor is simplified since the data is mapped into sequential locations in each memory module. A high I/O bandwidth can be obtained since all I/O transfers will be done using direct memory access. It is also possible to use a parallel head disk arrangement for secondary storage which will yield fast transfer times.

A possible modification of the data memory modules can be made. In this arrangement, each memory module consists of two submodules where it is possible to access data from only one submodule at a time. While operations are being performed on the data in one submodule, new data can be input to the other submodule or results of a previous process can be output. Thus, input/output can be overlapped with the computational process. This type of arrangement would be useful in a full production mode of operation. This mode would correspond to the situation where the same operations are being

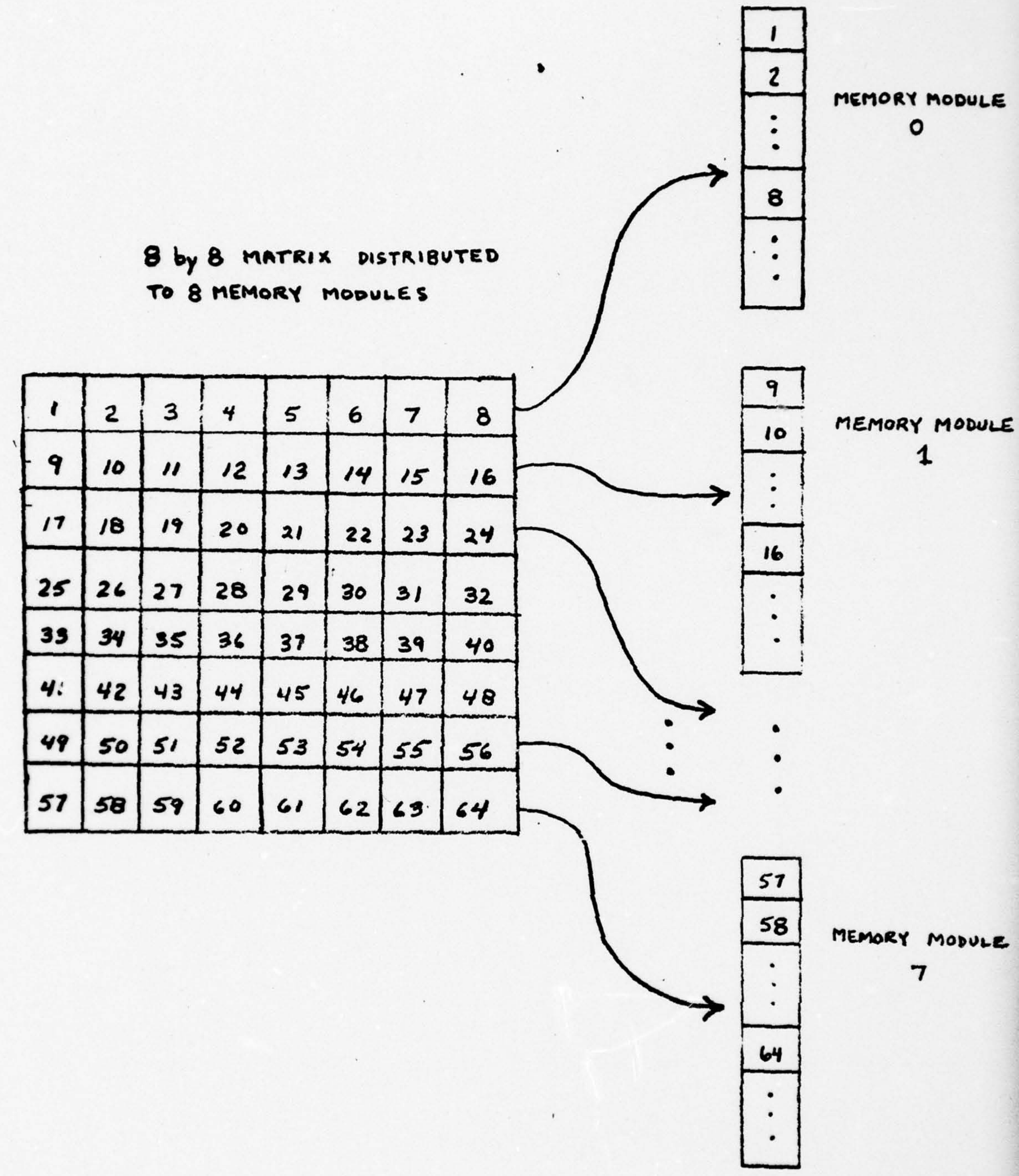


Figure (7) Example of data distributed to memory modules in now major form performed on many images of the same type and high throughput is needed. An example of this would be an automated white blood cell count system. In a research oriented environment, this modification would not be very useful. This is due to the highly experimental and interactive nature of the computations being performed.

The instruction memory is partitioned into four modules. Each module consists of memory along with sequencing hardware which is used to broadcast instructions to a partitioned set of microprocessors. These modules are loaded by the sequential controller through a single bus. If the system is to operate with a single group of N microprocessors, then all four modules are loaded in parallel with the same instructions. If two tasks are to be executed, each with N/2 microprocessors, then two modules are loaded with the instructions followed by the other two modules being loaded with the instructions for the other task. If four separate tasks are to be executed, then each module is loaded with a different task.

Memory Management and I/O Processor

The memory management and I/O processor (MMIO) is responsible for the allocation of data to each memory module, control of the direct memory access process, and all input/output interfaces. There are three modes of data allocation and transfer possible for the MMIO.

The first mode of operation consists of a sequential access to the memory modules. This mode is best described in terms of an example. Suppose data from a slow-scan (15 field/second) television camera scanner [16] is to be input to the system. It is assumed that the image is 256 by 256 pixels and that there are 64 microprocessors in the system. The MMIO processor will operate in the following manner. Upon command from the Sequential Controller, the MMIO processor first determines that 4 rows of 256 pixels each are to be allocated to each memory module. It then sets up a direct memory transfer for 1024 bytes. The memory locations where the 1024 bytes of information are to be stored is determined by the mode, the internal starting address, and the memory module number. The actual hardware implementation of this will be given following the description of the other modes. In the scanner, data is sampled and digitized on a line by line basis with 256 pixels obtained per horizontal sweep. The scanning process is initiated by the MMIO and coincides with the beginning of a new television field. This beginning of a field is denoted by a vertical synch pulse preceeding a horizontal synch pulse. Once 1024 pixels have been input to the memory module, the MMIO processor changes the address to the next memory module, sets up a block transfer of 1024 bytes, and reinitiates the scanning process upon receit of the next horizontal synch pulse. All processor changes are made during the horizontal retrace time of the slow-scan television camera, which amounts to approximately 50 microseconds. Therefore, it is possible to input a 256 by

256 image in 1/15 of a second, the time required to scan one television field.

The second mode of operation allows for the broadcasting of input data to partitioned subsets of the memory modules. In terms of the example just discussed, suppose the system was partitioned into two subsets of processors, then 8 rows of 256 pixels are allocated to each memory module. Then, in this mode of operation a block transfer of 2048 bytes would be made to memory module (i) and memory module (i+1) simultaneously, where i and i+1 represent the module number.

The third mode of operation allows for the parallel transfer of data between the memory modules and a parallel head disk. This type of transfer is similar to that used in the Illiac IV computer [17].

Figure (8) indicates a proposed hardware design that allows for the addressing of the memory modules under the three different modes of operation. All transfers of data under the first two modes occurs over a tri-state bus. In the third mode, all transfers occur over a multiplexed parallel bus. This system allows for the partitioning of the microprocessors and memory modules into one, two or four groups. The addressing of the memory modules is accomplished through the use of decoders, and decoder outputs are used as enable lines to the memory modules. Each mode of operation is encoded into a set of bits which are used to enable a particular decoder. Mode 2 has two possible bit representations, one for a two group partition and the other for a four group partition. Mode 3 also has two possible bit representations since special control is needed for the multiplexed parallel bus. One representation sets up the transfer from the memory modules to the parallel head disk, while the other sets up the transfer from the disk to the memory modules.

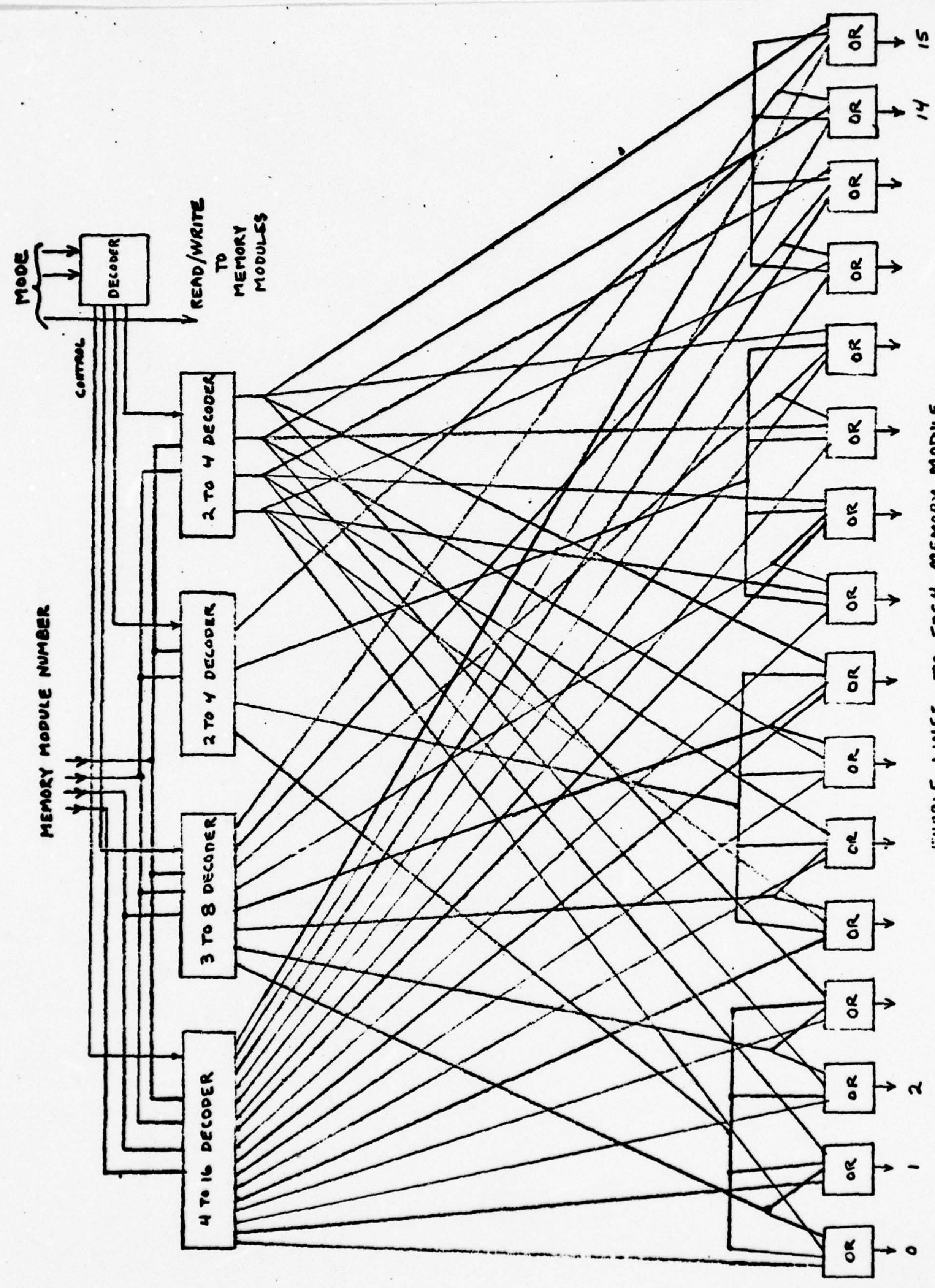
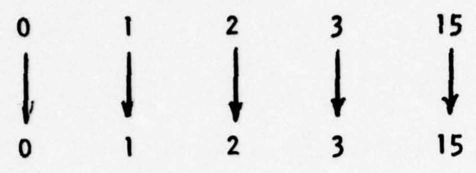


Figure (8) Diagram of MMIO Memory Module Selection Logic under Mode Control

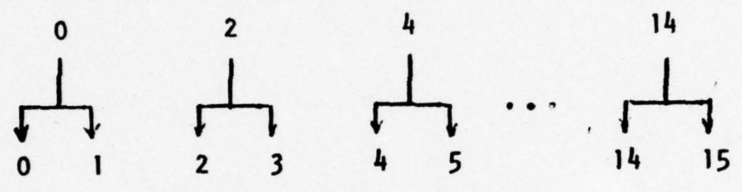
(continued on next page)

MODE 1 MEMORY MODULE NUMBER



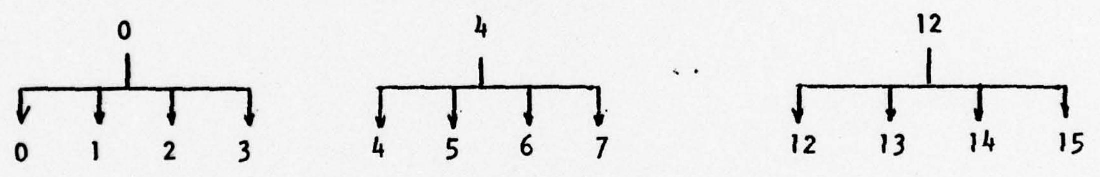
ENABLE LINE

MODE 2 (2 group) MEMORY MODULE NUMBER



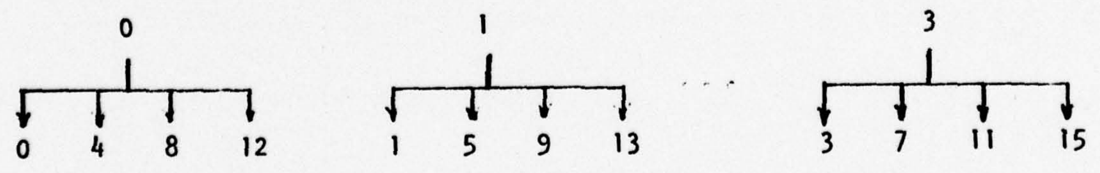
ENABLE LINES

(4 groups) MEMORY MODULE NUMBER



ENABLE LINES

MODE 3 (Assuming 4 head parallel disk) MODULE NUMBER



ENABLE LINES

Figure (8) - Continued -

Interconnection Network

The emphasis in the design of this system has been to provide the capability for use of Single Instruction Multiple Data or Multiple Instruction Multiple Data stream modes. This capability is a function of the versatility of the interconnection network.

The simpliest type of interconnection network is a single bus. In this structure, all microprocessors and memory modules communicate over the same pathway. The single bus connection is inexpensive to build and readily expandible, but has the disadvantage that only one microprocessor is allowed to send information at a time. This single bus then becomes a bottleneck in system performance.

On the other end of the scale, the crossbar switch ranks as the most versitile interconnection scheme. It allows each microprocessor to connect to any memory module as long as that module is not already connected to another processor. The cost and complexity of this scheme is prohibitive, since O(N**2) switches are required to construct such a system, when there are N microprocessors and N memory modules in the system.

It is clear that a practical interconnection network must have many of the capabilities of a crossbar switch without the enormous cost. Several interconnection networks that fit into this intermediate category have been discussed by Siegel [18]. These include the Perfect Shuffle, Cube, Illiac, Plus-minus 2**i (PM2I), and Wrap-around Plus-minus 2**i (WPM2I). Parallel processing systems that incorporate some of these interconnection networks have been constructed [19,20,21].

Each microprocessor has a unique integer in the range 0 to N-1 associated with it which serves as its address. Similarly, the memory modules can be addressed by the same range of integer values. Each microprocessor is allowed to directly access only a subset of the N memory modules. The path between a microprocessor and a memory module is chosen as a function of the microprocessor's address bits. Each network has a different function set of the address bits to choose the interconnection paths. For example, the Plus-minus 2**i network can be represented by the following 2m functions:

$t_{+i}(j)= j+2**i \bmod N$

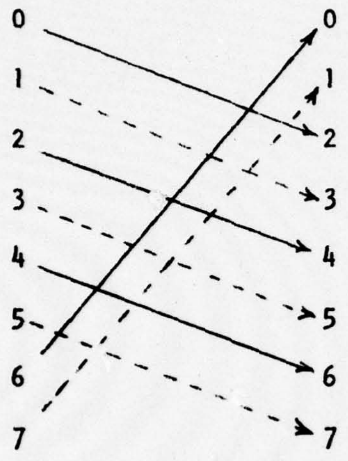
$t_{-i}(j)= j-2**i \bmod N \qquad \text{for } 0<= i < m$

where $m = \log_2 N$ and j is the address of the microprocessor. This interconnection network allows access to an arbitrary memory module in at most m steps.

As mentioned before, the interconnection scheme must have the capability to partition the microprocessors and memory modules into subsets so multiple instruction streams can be implemented. The Plus-minus 2**i network is a possible canidate since it allows partitioning of the microprocessors and memory modules. For example, as shown in Figure (9), if only functions t_{+1}, t_{+2}, t_{-1}, and t_{-2} are used, the microprocessors and memory modules are partitioned into two separate groups. If only the functions t_{+2} and t_{-2} are used, four seoarate groups are formed.

Since this entire system is being structured for image processing use, certain image processing features must also be considered in the choice of the interconnection network. For example, suppose each memory module contains only one row of the picture matrix. In order for microprocessor (i) to perform a local operation based on a 3 by 3 pixel window, it must directly access memory modules i, i-1, and i+1 to take full advantage of the potential throughput of the system. When a 5 by 5 window is used, direct access to memory modules i-2 and i+2 would also be needed. The PM2I and WPM2I interconnection networks are the only ones presently suggested which allow

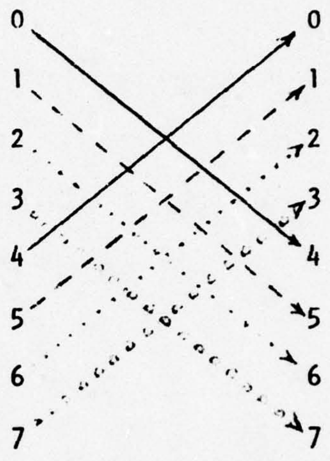
FOR N=8



if $t \pm 1$ and $t \pm 2$

Cyclic group 1: 0, 2, 4, 6

Cyclic group 2: 1, 3, 5, 7



if $t \pm 2$

Cyclic group 1: 0, 4

Cyclic group 2: 1, 5

Cyclic group 3: 2, 6

Cyclic group 4: 3, 7

Figure (9) Partitions available in the PM2I intercommenction network direct accesses of this form. However, it is not possible to partition the WPM2I network.

Another computation that is sometimes used in image processing is the Fast Fourier Transform (FFT). Pease [22] and Stone [23] have shown how to compute this algorithm in parallel using a Perfect Shuffle. This computation is accomplished through a sequence of operations performed first on pairs of numbers whose binary representation of their indices differ by 2^{m-1}, then on those that differ by 2^{m-2}, and so forth, to those that differ by 2^0. The Cube and PM2I interconnection networks allow the same pairing of data, so it is possible to compute this algorithm directly on any system using one of these interconnection networks.

Since the PM2I network fulfills the data manipulation requirements, it will be used in the design of the system. Its implementation will differ from previous designs. For example, Feng's data manipulator [19] was implemented using $\log_2 N$ stages of PM2I functions. The implementation that will be used in the system will be a recirculating stage. One pass through this stage will allow a microprocessor to access a memory module with any of the following addresses; j, $j+2^0$, $j-2^0$, $j+2^1$, $j-2^1$,..., $j+2^{m-1}$, where j is the microprocessor address. Access to an arbitrary memory module can occur after at most m recirculations through the stage. Each microprocessor will have a recirculation buffer to facilate such transfers.

Figure (10) shows a possible hardware implementation of the recirculation stage connected to a microprocessor. Only one bit of the data, address and control buses is shown. Tri-state buses are used extensively.

Control over the interconnection network is viewed on two levels, a logical level and a physical level. When the system is partitioned into only one group, both levels coincide. If partitioned into two groups the func-

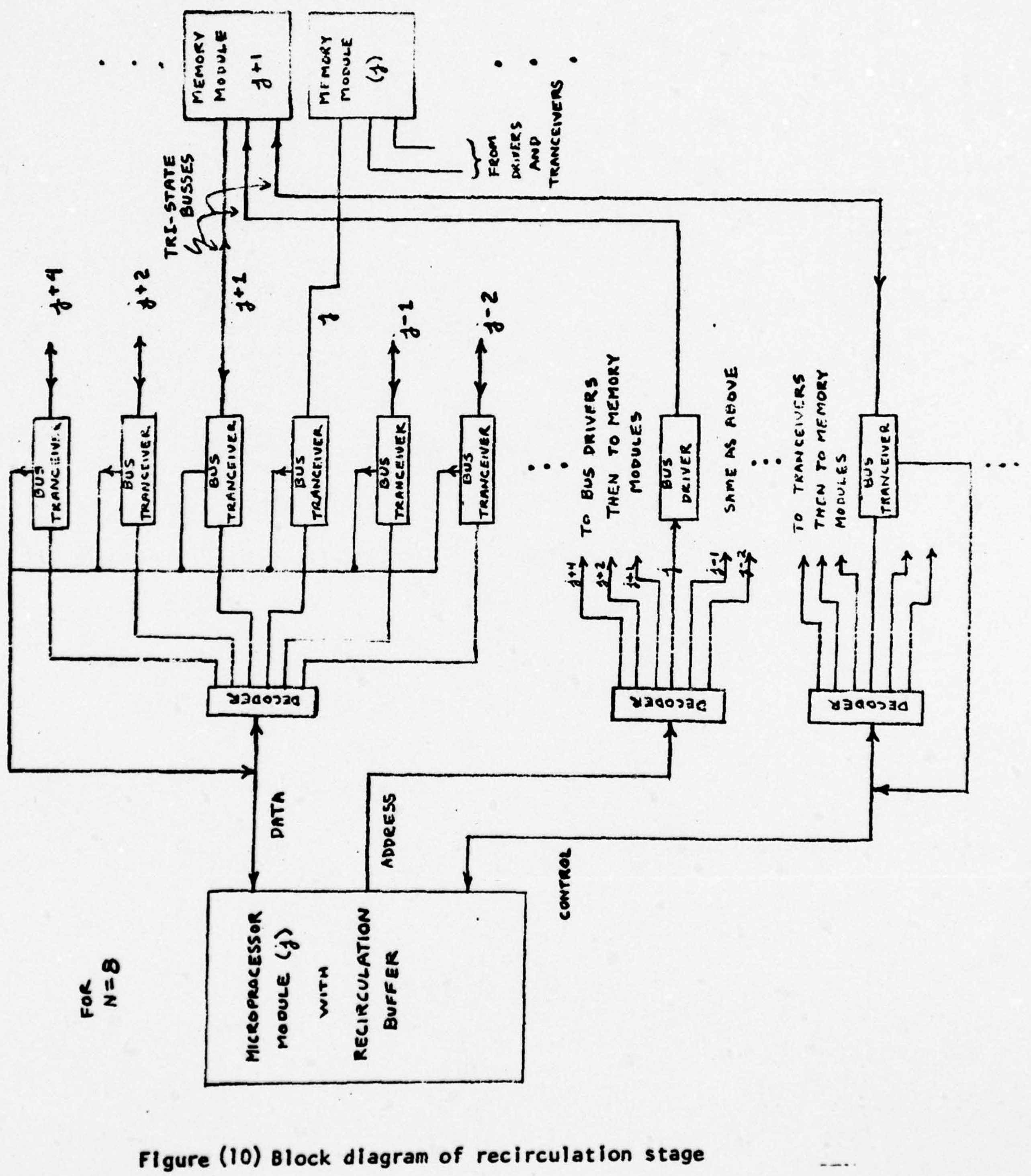

FOR N=8
MICROPROCESSOR MODULE (j) WITH RECIRCULATION BUFFER
DATA
ADDRESS
CONTROL
DECODER
DECODER
DECODER
BUS TRANCEIVER
BUS DRIVER
j+4
j+2
j+1
j
j-1
j-2
TRI-STATE BUSSES
MEMORY MODULE j+1
MEMORY MODULE (j)
FROM DRIVERS AND TRANCEIVERS
TO BUS DRIVERS THEN TO MEMORY MODULES
SAME AS ABOVE
TO TRANCEIVERS THEN TO MEMORY MODULES

Figure (10) Block diagram of recirculation stage tions t_{+0} and t_{-0} are not allowed. However, on a logical level we would like to use these functions to provide a consistent view of the interconnection network in terms of our algorithms. Thus, the algorithms become partition independent. The translation between the logical level and the physical level can be accomplished in two possible ways. One way would be during language compilation. An interconnection program control statement would be compiled based on the global partition information. Another possible implementation of the control would be through a conditional branch in the microcode which is set up by the partition information.

Microprocessor

In the context of this report, the term microprocessor refers to a processing module which is built out of a set of LSI chips that are used as building blocks in its construction. Each chip is usually a 2 or 4 bit wide section or slice of a functional unit which allows cascading to form systems with word lengths of up to 64 bits. Coordinated control of these chips is through user microprogramming. The use of this type of microprocessor instead of more conventional versions, such as Intel's 8080 or Motorola's M6800, is based on a number of factors.

These factors include:

(1) high processing speed,

(2) capability to optimize the system instruction set for a particular application, and

(3) capability to upgrade system through addition of new features or better performance by modification of the microcode.

Some disadvantages of using these bit slice microprocessors are:

(1) higher cost,

(2) very little software support, and

(3) higher power consumption.

The high processing speed is based on the type of technology used in their construction and the minimization of gate propagation delays due to the high packing density of logic gates per chip. In Figure (11), the throughput capability of various microprocessors using a particular instruction mix is compared with a missle guidance and control computer system [24] based on the AM2901A bit slice microprocessor designed by Advanced Micro Devices [25]. The throughput, which is measured in thousands of operations per second (KOPS), is at least an order of magnitude greater than its

Throughput Capability of
Various Microprocessors
Using an Instruction Mix
of: 50% Memory Ref.
30% Register Ref.
15% Multiply
5% I/O

Microprocessor	Word Size	Throughput (KOPS)
8080A	8	5
6800	8	5
PACE	16	25
IMP-16	16	34
TMS 9900	16	73
MICRONOVA	16	79
System Design Using AM 2900 Bit Slice Microprocessor	16	808

Figure (11) Throughput capability of various microprocessors nearest competitor.

The following is a list of currently available bit slice microprocessors. Some of the bit slice microprocessors are organized around a family of support chips.

Intel 3000 series family [26] - Intel Corporation

MMI 6700 series family [27,28] - Monolithic Memories Incorporated

AM2900 series family [25] - Advanced Micro Devices

SN74S481, SBP0400A and SBP0401A [29] - Texas Instrument

M10800 series family [30] - Motorola

A brief description of each bit slice microprocessor and their associated microprogram control unit, if it exists, will be given.

The Intel 3002 is the central processing element (CPE) of the family. It is a two bit slice. Each CPE (Figure (12a)) is organized with five independent busses, three for input, two for output. A sixth bus is used for control of the CPE. This control allows the performing of over 40 boolean and binary functions. The CPE has a cycle time of 150 nanoseconds.

The Intel 3001 (Figure (12b)) is the microprogram control unit. The address capabilities of the 3001 (MCU) are unique. Microprogram addresses are organized as a two-dimensional array or matrix. A 9 bit address specifies the row address with the upper 5 bits and the column address with the lower 4 bits. From a particular row or column address, it is possible to jump either unconditionally to any location in that row or column or conditionally to a specified subset of locations, in one operation. The MCU has a cycle time of 700 nanoseconds. For a 16 bit system with a pipelined architecture, a microinstruction cycle time of 150-200 nanoseconds can be obtained.

The MMI 6701 (Figure (13a)) is a 4 bit Schottky LSI microprocessor slice. Thirty-six microinstructions are used to control arithmetic, logical,

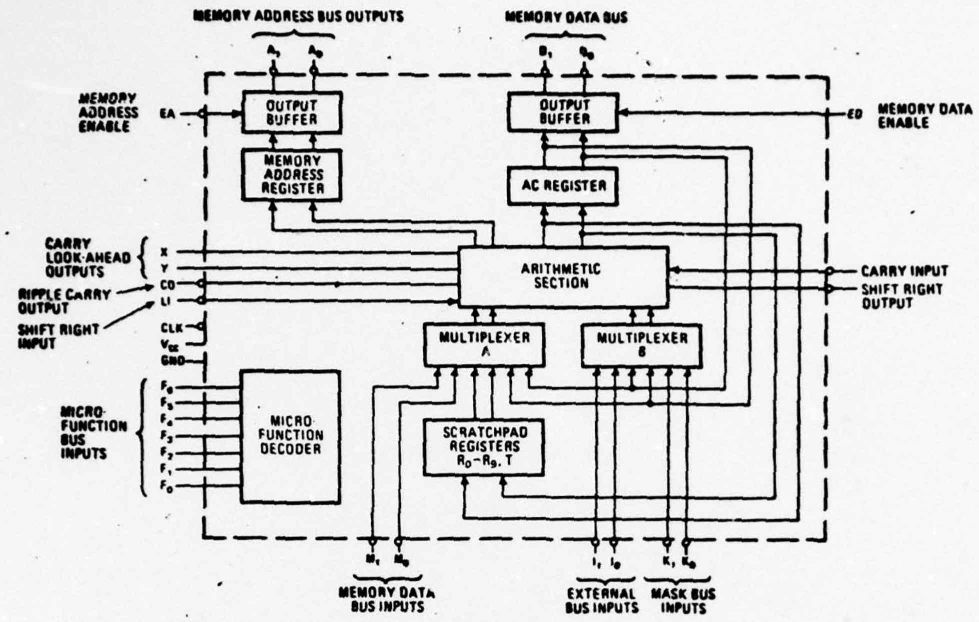


Figure (12a) INTEL 3002 two bit slice Central Processing Element (CPE)

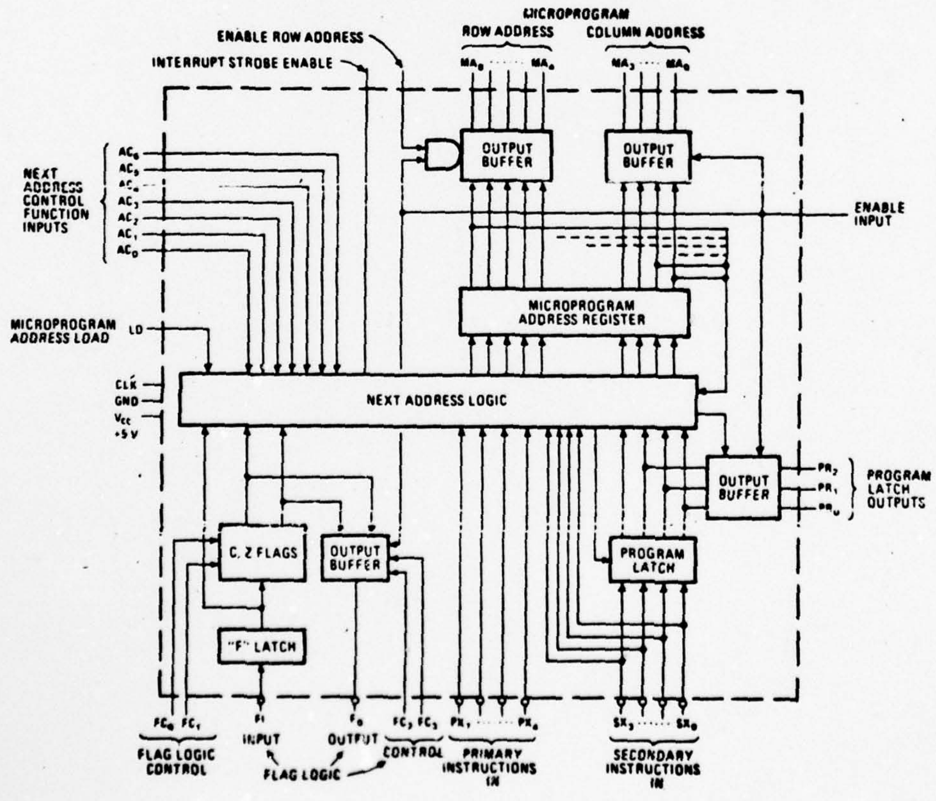


Figure (12b) INTEL 3001 Microprogram Control Unit (MCU) and shifting operations. Overflow detection is provided. The microprocessor slice can work with either positive or negative logic. There are 16 directly addressable, two port, general purpose accumulators with two address capabilities. The two address capability allows for working on two accumulators at once. Three register operations are available. The Q register is used as either a scratchpad or accumulator extension. The MMI 6701 has a cycle time of 200 nanoseconds.

The MMI 6700 (Figure (13b)) is the microprogram controller (MC) or sequencer. The MC can be used with RAM, ROM, or PROM and can directly address up to 512 words of cntrol memory. A number of instructions are available for conditional and subroutine jumps. A multi-way branching capability at each microinstruction is provided. The MC has in addition to a single level of subroutine, a control counter which allows the repetition of microinstructions. In a system without pipelining, the microinstruction cycle time is approximately 250 nanoseconds.

In the AM2900 family, there are currently two microprocessor slices available, the AM2901 and the AM2901A. The AM2901A is a 20-30% faster version of the AM2901. The AM2901 (Figure (14a)) and the MMI 6701 microprocessor slices are very similar in organization. Each have the same register and Q register organization. Both allow two operands to be read from the register file, have an operation performed in the ALU, shifted and written back into the register file during one microinstruction time. In parallel with this, the ALU output and Q registers can be right shifted. The main advantage of the AM2901 over the MMI 6701 is speed. It has a cycle time of 105 nanoseconds. The AM2901A has a cycle time of 70 nanoseconds.

The AM2909 and AM2911 are microprogram sequencers (Figure(14b)). They are 4 bit slices which are cascadable. The AM2909 can select an address from

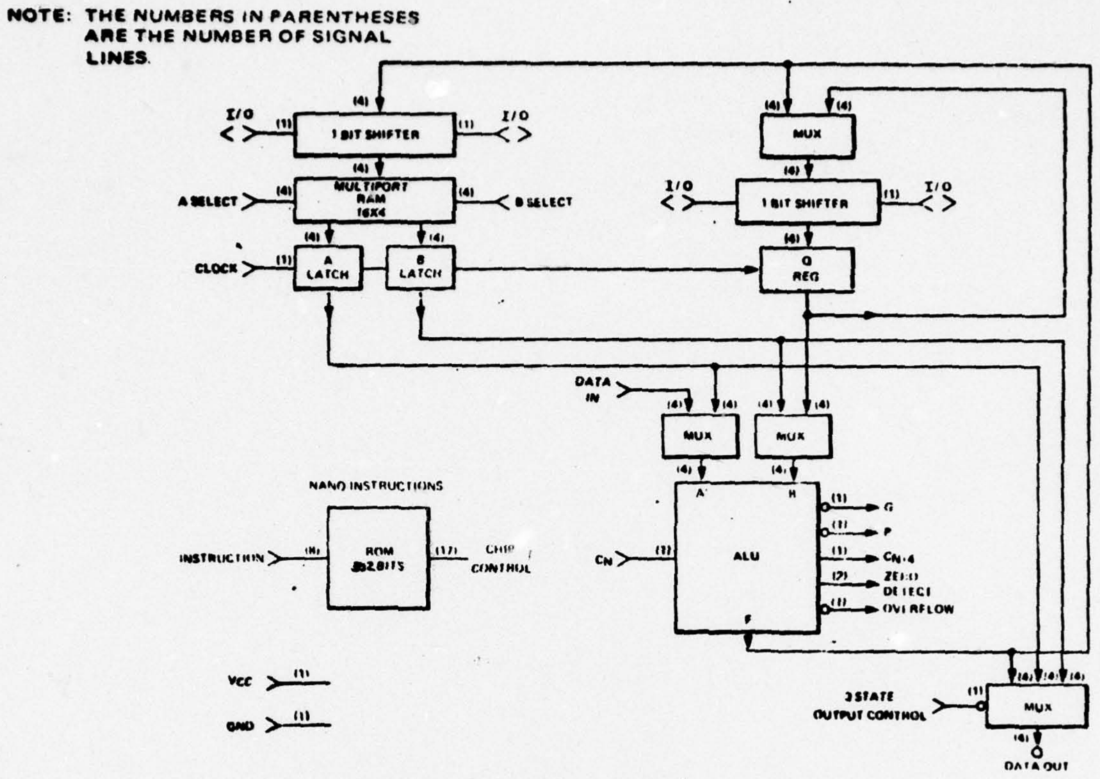


Figure (13a) MMI 6701 four bit microprocessor slice

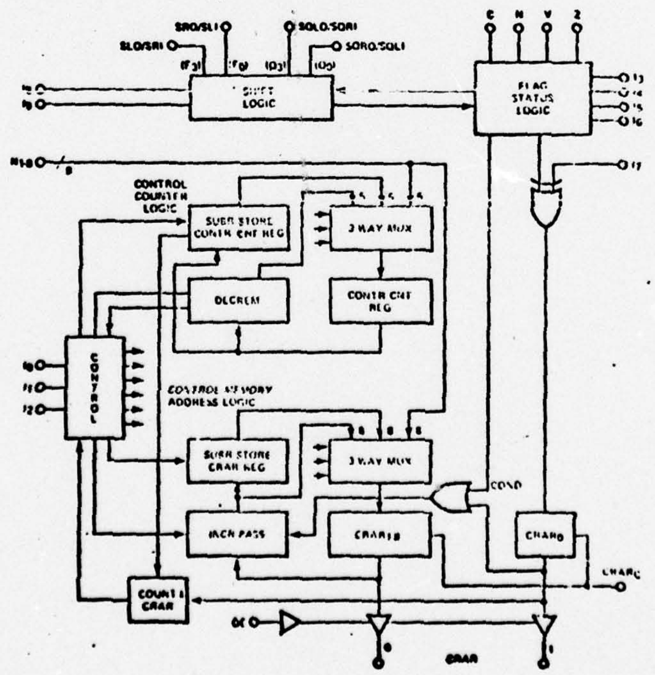


Figure (13b) MMI 6700 Microprogram Controller (MC)

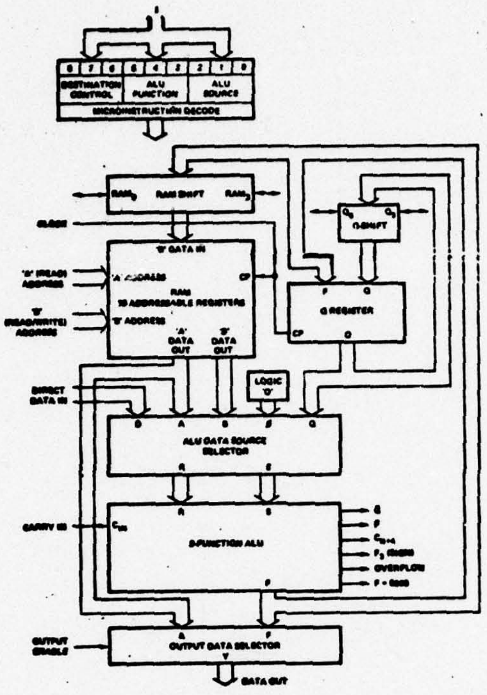


Figure (14a) AM 2901 four bit microprocessor slice

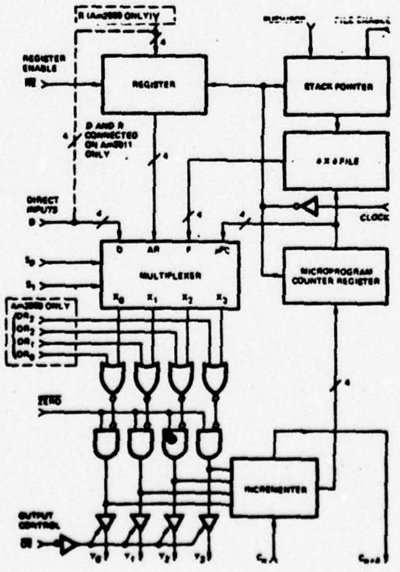


Figure (14b) AM 2909/AM 2911 microprogram sequences

four possible sources. These are a set of external direct inputs (D); external data from the R input, stored in an internal register; a four word deep push/pop subroutine stack; or a program counter register. Each of the four outputs can be OR'ed with an external input for conditional skip or branch instructions, and a separate line can force the outputs to all zeros.

The AM2911 is identical to the AM2909 except the OR gates are removed and the D and R inputs are tied together.

The microinstruction cycle time for a system built up in a pipelined fashison, should be approximately 100 nanoseconds.

A new 4 bit microprocessor slice, the AM2903 (Figure(15)), should become available in November, 1977. The AM2903 will be able to perform the same functions of the AM2901A and also provide powerful enhancements for use in arithmetic-oriented processors. It will have an infinitely expandable memory and three port, three-address architecture. The AM2903 has built-in multiplication, division and normalization logic along with parity generation and sign extension circuitry. This will allow easy implementation of multiplication, division, normalization of floating point numbers and other previously time-consuming operations.

The SN74S481 (Figure (16a)) is a 4 bit expandable Schottky microprocessor slice. Some of its architectural features include parallel dual input/output ports, full-function ALU with carry look-ahead, magnitude, and overflow decision capabilities, dual memory address generators, and double-length accumulator with shifting and sign-bit handling capabilities. Asynchronous access to data routing and counter updating controls is provided. In a single microinstruction, it is possible to perform an ALU function with a shift, select destination with address/iteration updating, plus address and present data to memory. Pre-programmed multiply, divide, and CRG algo-

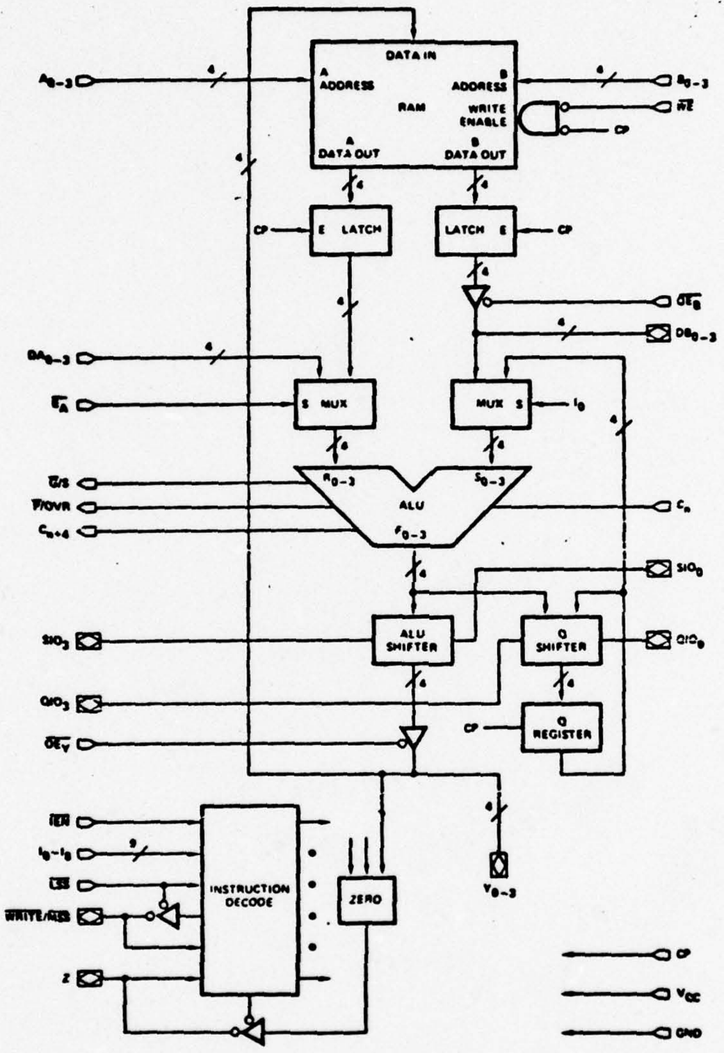


Figure (15) AM2903 four bit slice microprocessor

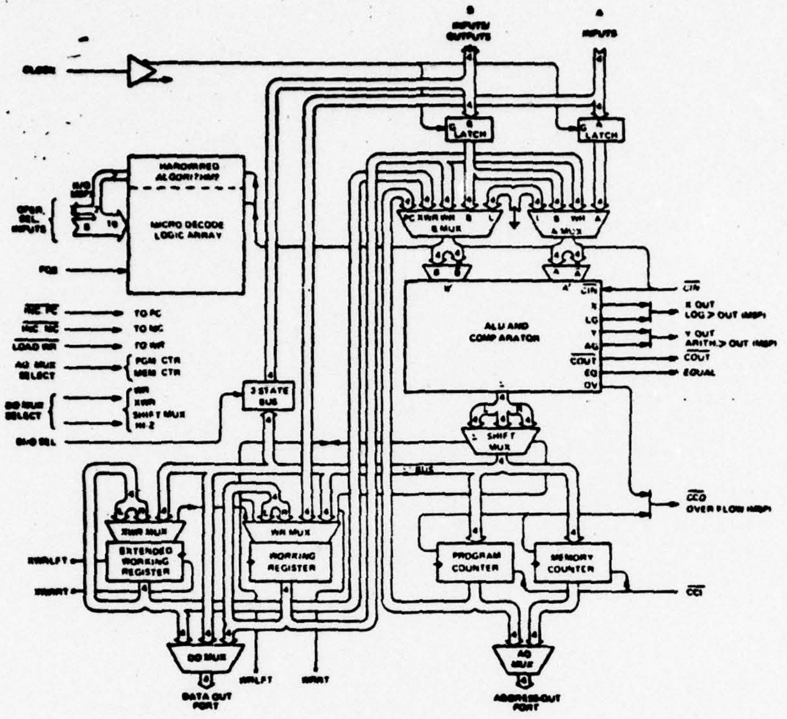


Figure (16a) Texas Instrument SN74S481 four bit slice Schottky processor element

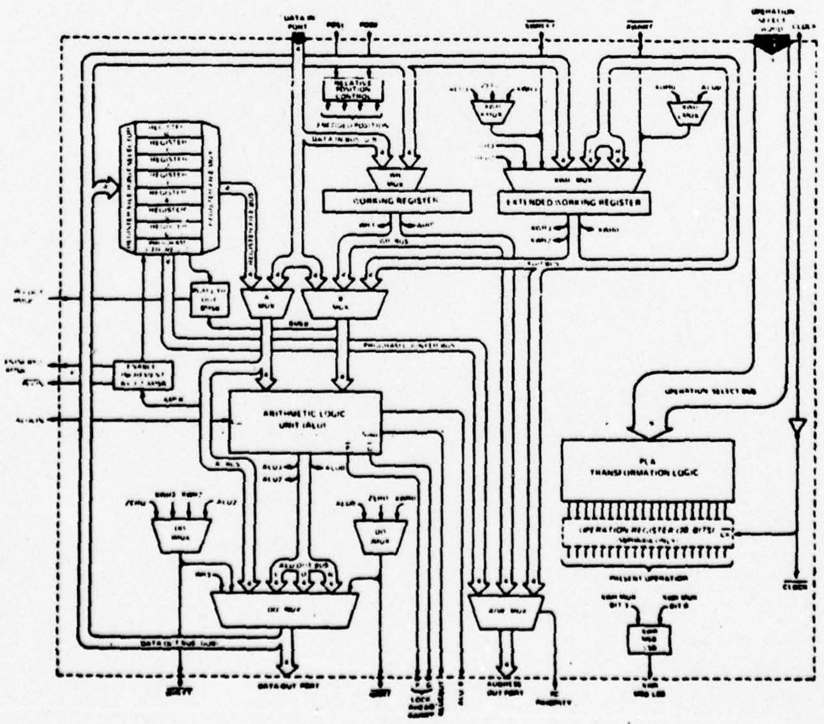


Figure (16b) Texas Instrument SBPØ4ØØA/SBPØ4Ø1A

rithms are provided. The microinstruction time is 100 nanoseconds.

The SBP0400A and SBP0401A (Figure (16b)) are 4 bit expandable microprocessor slices built out of Integrated Injection Logic (I^2L). Each have separate data-in, data-out, address-out and control ports. Sixteen functions are provided in the ALU along with a carry look-ahead capability. Some other features are 8 general registers including a program counter with independent incrementer, two working registers and shifters with on-chip handling of end conditions. The major difference between the two microprocessor slices is the SBP0400A has an on-chip pipeline operation register while the SBP0401A is designed for use in an externally pipelined system.

Both processors have a wide performance range. A constant speed-power product can be obtained over an injector current range covering three oorders of magnitude with a typical ALU/shift operation of 240 nanoseconds at 200 mW nominal power.

The M10800 family is built out of Emitter Coupled Logic (ECL). The MC10800 (Figure (17)) is a cascadable 4 bit ALU slice. This chip can perform logic operations, binary and BCD arithmetic, and both logic and arithmetic shifting. An internal accumulator is available for temporary storage. A special mask network allows bit masking of data before arriving at the ALU. Three independent data ports are provided. Two ports are input/output while the other is input only. The following arithmetic and status outputs are provided: overflow, sign, zero, carry out, group propagate, group generate, parity of carries and parity of results.

The MC10801 (Figure (18)) is used for the microprogram control function. It is a 4 bit cascadable slice. A status register, instruction register, 4 level subroutine stack, address register, retry register and incrementer are included on the chip. Sixteen instructions are available for use

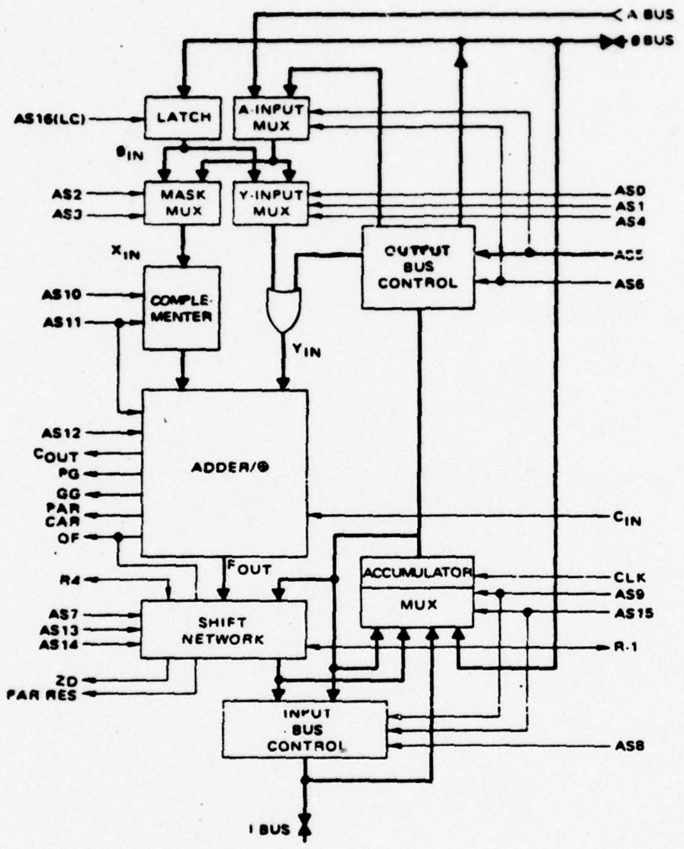


Figure (17) Motorola MC 10800 four bit ALU slice

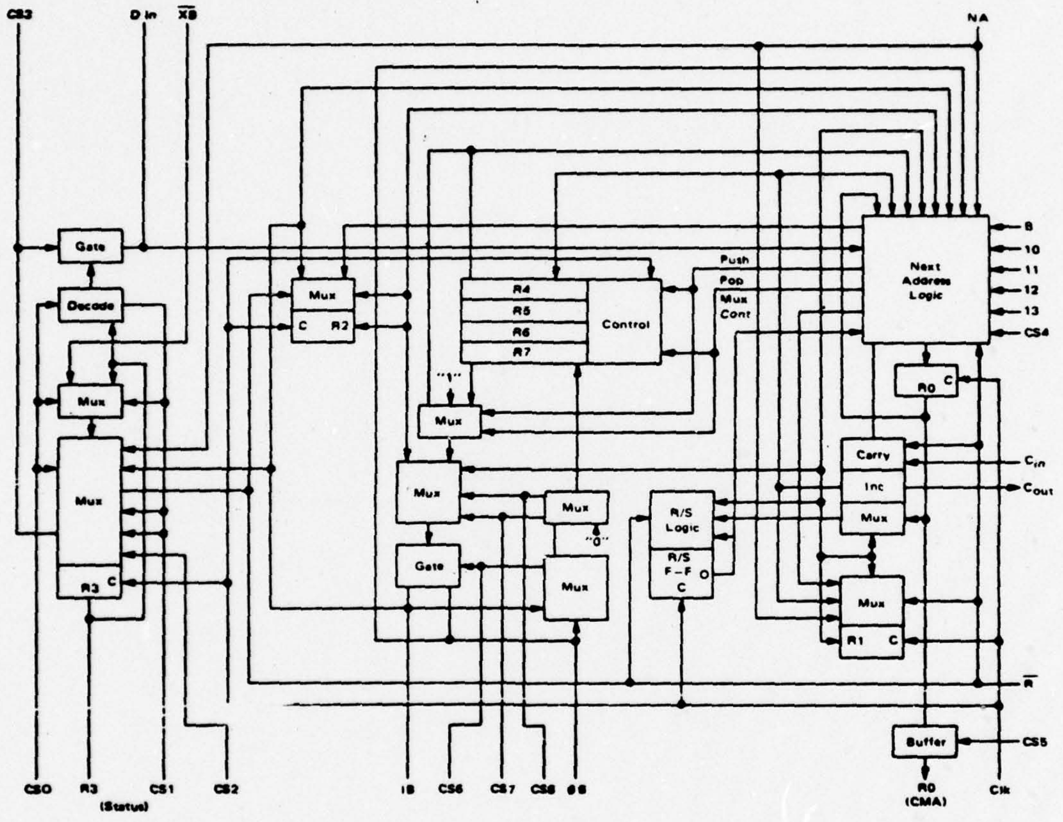


Figure (18) Motorola MC 10801 four bit slice microprogram control in generating the next control memory address. Some of these instructions include increment, direct jumps to various inputs and registers, subrouting, conditional jumps, and a special instruction for multipath branching.

The MC10803 (Figure (19)) provides the memory interface function. This device has six registers for uses as memory address register, memory data register, program counter, stack pointer, index register, or other functions. Seventeen data transfer instructions are provided. Since an ALU is included, memory address generation under various addressing modes, can take place completely on this chip.

The MC10808 (Figure (20)) is a bit programmable multi-bit shifter. This device provides a very fast shift network that is essential in floating point operations for prenormalization or alignment of exponents.

The combination of the above devices into a system can provide microinstruction times of less than 100 nanoseconds depending on the system architecture and maximum path delay.

In order to get an estimate of the potential power of a multi-microprocessor system, it would be useful to compare the processing power of these bit slice microprocessors to that of a computer like the PDP 11/40 or PDP 11/70. A study was made at Carnigie Mellon University [31] where a complete equivalent PDP 11/40 was constructed using Intel's 3000 series bit slice family. The results of that study indicated that the processing speed of the equivalent system was 63% the speed of the PDP 11/40. A careful investigation of the performance showed a number of pitfalls due to the choice of Intel's 3000 series bit slices in the implementation of the system. If the system had been rebuilt using the AM2900 series bit slice microprocessor, the integrated circuit package count could be reduced from 144 to 95 and the overall performance boosted to the level of a PDP 11/40.

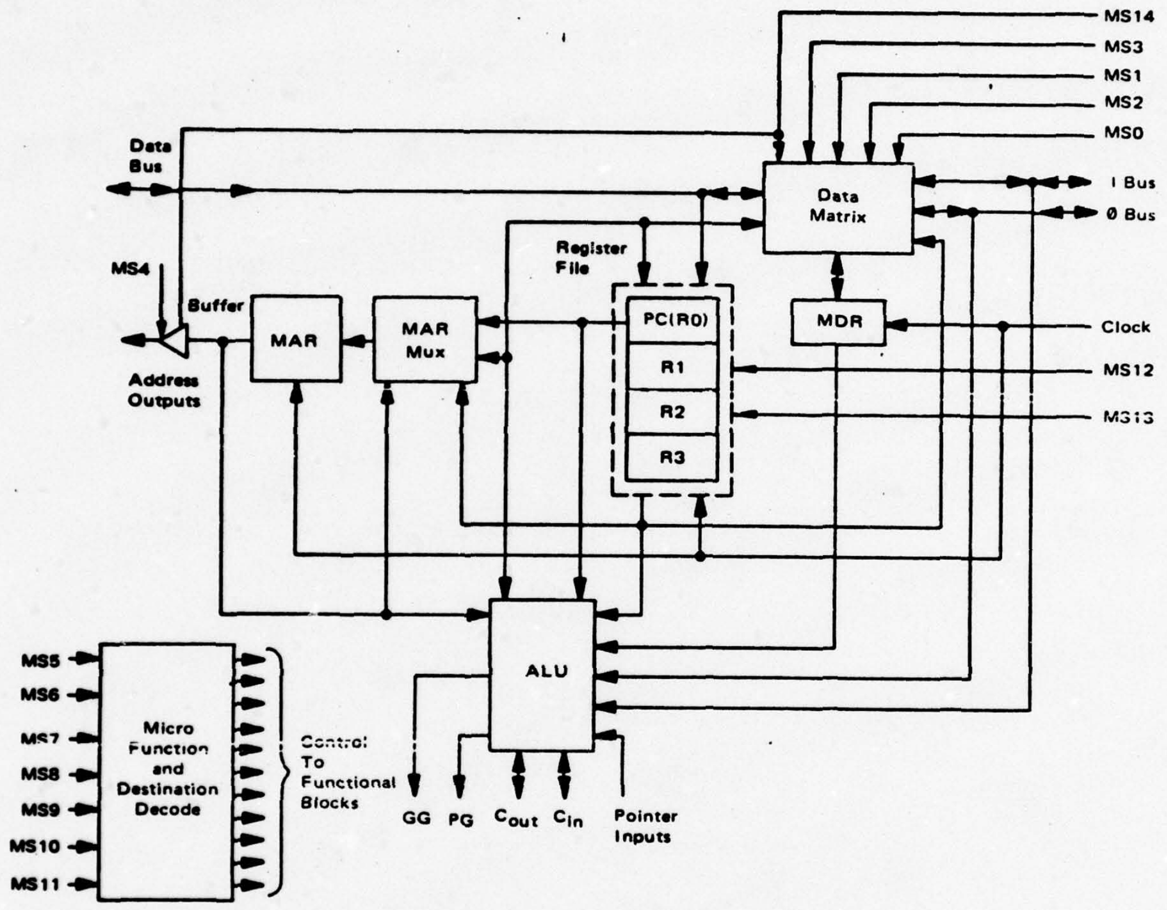


Figure (19) Motorola MC 10803 memory interface unit

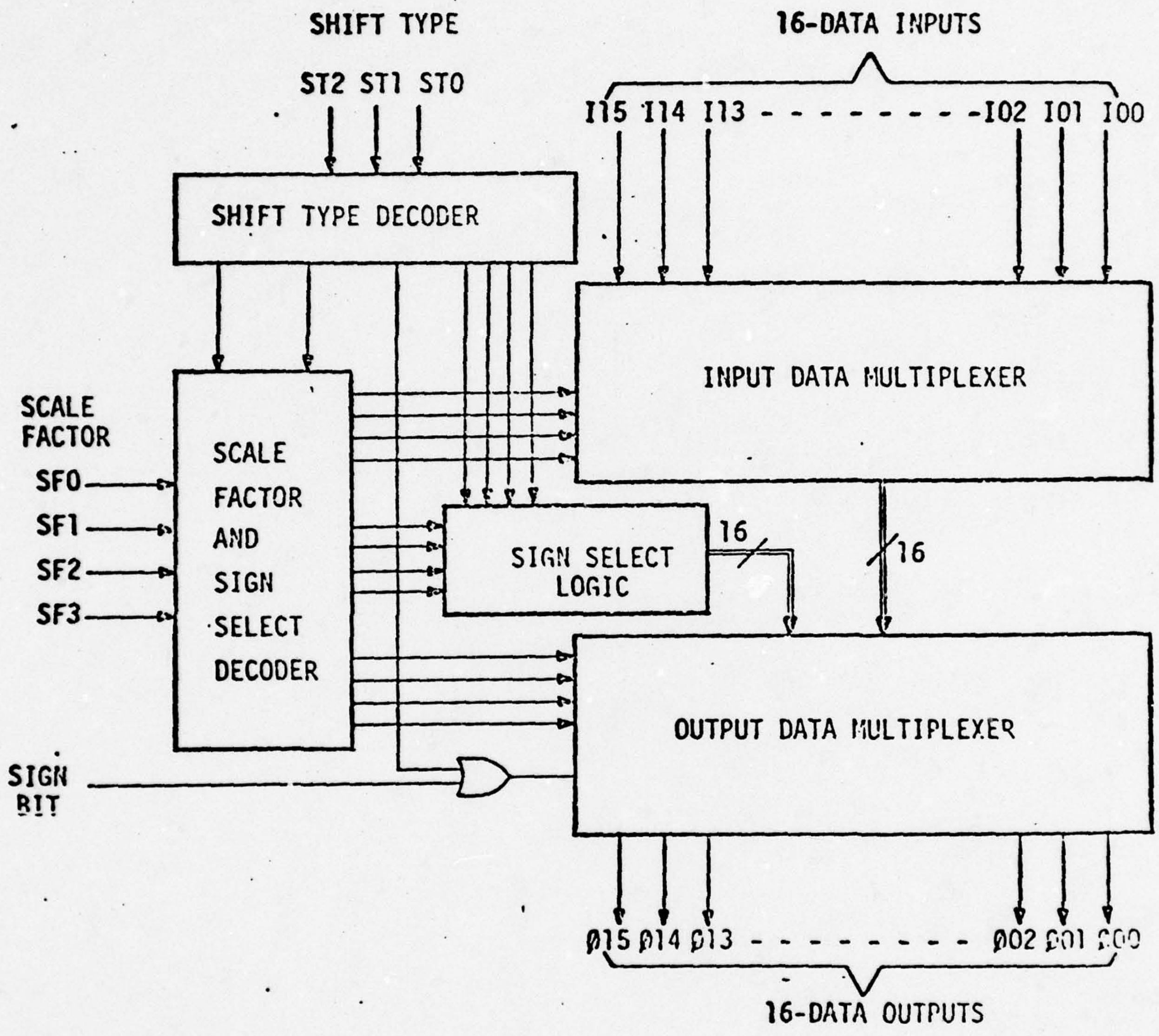


Figure (20) Motorola MC 10808 16 bit programmable multi-bit shifter

Sequential Controller

The sequential controller is used for program development and coordination of the multi-microprocessor system. Since its operation is fairly conventional, it will be a convential computer system, like the PDP 11/45.

A job control language will be used to specify commands to the various subsystems of the multi-microprocessor system. An example of two possible commands are "DISPLAY IMAGE (.)" and "PARTITION (.)" where "(.)" represents an arguement list.

The DISPLAY IMAGE command with the appropriate arguements would be sent by the Sequential Controller to the Memory Management and I/O processor for decoding. Once decoded, the MMIO would supervise the display of an image with respect to the given parameters. The PARTITION command would serve as a global control over the partitioning of the data memory modules, interconnection network, and instruction memory modules. These two commands are but two of the many possible commands the system will use.

In addition to the coordination activities of the Sequential Controller, it is also used for program generation, compilation, and instruction memory loading. Loading of the instruction modules takes place through a direct memory access channel.

The Sequential Controller is not used for any image processing computations, since it would become a potential bottleneck in the system operation.

Chapter V. Evaluation Study

A two part investigation has begun to estimate the execution time required for various image processing tasks. The first part of the study involves implementation of the tasks on a PDP 11/70 computer and comparing these results with those obtained for a single bit slice microprocessor system. The comparison is based on the total time required to complete the task as a function of the size of the image. Some preliminary results have been obtained for image processing tasks that operate on local neighborhoods. A program has been written that spatially filters or smoothes image data. Appendix I contains the program listing, along with a timing equation derived from the program. Using the instruction set of the previousily mentioned missile guidance computer system, a timing equation was obtained for a program which accomplishes the same task. Since the basic instructions are very similar to those of an HP 2116 computer [32] and a program listing of the same task was available, the problem of developing the timing equation was simplified. In Appendix I, the instruction list along with the execution time for each instruction is given. Also included is the program listing along with the timing equation obtained.

Table (2) shows the processing time required under both systems using various image sizes. A comparison of these results shows that the system built using the AM2900 series bit slice microprocessor is approximately 29% slower than the PDP 11/70. It is estimated that a similar performance efficiency can be obtained for operations such as edge enhancement, thinning, and other computations based on local neighborhoods.

The second part of this investigation is to determine how these same algorithms can be implemented on the multi-microprocessor system. Since the

Execution Times of Smoothing Program for Square Matrices of Side M

M	Time (Seconds) PDP 11/70	Time (Seconds) Microprocessor
64	.086	.124
128	.352	.495
256	1.43	1.98
512	5.74	7.91
1024	23.04	31.67

AM 2900 Microprocessor System is 28.6% Slower Than PDP 11/70

Table (2) Comparison of execution times for smoothing program exact structure of each microprocessor module has not yet been determined, this work will help in determining a structure which optimizes computation time.

As discussed in Chapter IV, the ability to compute parallel FFTs and to use local neighborhood algorithms resulted in the choice of the PM2I interconnection network for this system.

Investigations into the structure of the microprocessor modules have not yielded any specific results, and have, in fact, raised many more new questions. For example, the amount of local storage and the size of the recirculation buffer must be determined. Is floating point hardware needed? What type of performance gain is possible if automatic hardware address generation [33] is used? Finally and perhaps the most important question that must be answered deals with the type of language support the system should have. How should control of the interconnection network, active-inactive status [34] of the microprocessor modules, and data transfers be specified in this language? Also, since the system will be microprogrammed, should the language support a dynamic microprogramming capability? Much further research is needed into these areas before an optimized structure can be developed for this subsystem.

Chapter VI. Conclusions and Future Work

In this report, we have attempted to show how image processing differs from conventional computer processing and why a specialized computer architecture is needed for it. The use of cellular logic arrays for image processing was discussed. Although cellular logic arrays can be computationally very powerful, algorithm construction is very difficult, input/output is slow, and it is only feasible, at the present time, to build small arrays. Another problem is system reliability. Since these systems have not been designed to isolate non-functioning parts of the array, if one or more cells are inoperative, the logic array can produce completely erroneous results.

Another approach to the solution of the image processing computation was introduced in the remainder of this report. The partitionable multi-microprogrammable microprocessor system is an attempt, using current technology, to provide a reliable, flexible, and easy to use system for image processing.

The overall architecture of this system was heavily influenced by the computational requirements of various image processing tasks. The system should attain a high degree of reliability since it is partitionable. If a failure occurs, the same task can be executed on a smaller partition, by bypassing the malfunctioning device or devices. It was shown that bit slice microprocessors are needed to provide sufficient system flexibility and computational power. In order for this system to be easy to use and efficient, considerable research is needed in the development of a higher level language that can be mapped onto the system hardware. In the past, most computer systems were designed first, with development of a higher level language as an afterthought. This has proved to be disasterious in the cases of the Illiac IV and Staran computers. The Burroughs B5700/6700 series computers [34] on the other hand are examples of systems whose architecture is consistent and facilitates the execution of algorithms written in a high level language, Extended Algol 60. In this system, a similar unified approach will be taken where the architecture and software are jointly developed. Further work must be done in determining the computational requirement of various image processing algorithms and how they can be implemented on this system. Also, the actions and interactions of the various subsystems must be simulated to determine global weakness or deficiencies in this system.

Appendix I

The execution time [36] for an instruction on the PDP 11/70 depends on the instruction itself, the modes of addressing used, and the type of memory.

The basic instruction set timing is:

Double Operand

all instructions,
except MOV: Instr. Time = SRC Time + DST Time + EF Time
MOV instruction: Instr. Time = SRC + EF

Single Operand

all instructions: Instr. Time = DST Time + EF Time or
Instr. Time = SRC Time + EF Time

Branch, Jump, Control, Trap and Misc.

all instructions: Instr Time = EF Time

The following charts are used in determining the execution time for an instruction.

C.1.5 SOURCE ADDRESS TIME

Instruction	Source Mode	SRC Time	Read Memory Cycles
Double Operand	0	.00	0
	1	.30	1
	2	.30	1
	3	.75	2
	4	.45	1
	5	.90	2
	6	.60	2
	7	1.05	3

C.1.6 DESTINATION ADDRESS TIME

Instruction	DST Mode	DST Time (A)	Read Memory Cycles
Single Operand and Double Operand (except MOV, MTPI, MTPD, JMP, JRS	0	.00	0
	1	.30	1
	2	.30	1
	3	.75	2
	4	.45	1
	5	.90	2
	6	.60	2
	7	1.05	3

NOTE (A): Add .15 μsec for odd byte instructions, except DST Mode 0.

C.1.7 EXECUTE, FETCH TIME

Double Operand

Instruction (Use with SRC Time and DST Time)	EF Time (SRC Mode 0) (DST Mode 0)	Read Mem Cyc	EF Time (SRC Mode 1-7) (DST Mode 0)	Read Mem Cyc	EF Time (SRC Mode 0-7) (DST Mode 1-7)	Read Mem Cyc
ADD, SUB, BIC, BIS MOVB	.30 (D)	1	.45 (D)	2	1.20 (C)	1
CMP, BIT	.30 (D)	1	.45 (D)	1	.45 (C)	1
XOR	.30 (D)	1	.30 (D)	1	1.20	1

NOTE (C): Add 0.15 μsec if SRC is R1 to R7 and DST is R6 or R7.
NOTE (D): Add 0.3 μsec if DST is R7.

Instruction (Use with SRC Time)	DST Mode	DST Register	EF Time (SRC Mode = 0)	EF Time (SRC Mode = 1-7)	Read Memory Cycles
MOV	0	0-6	.30	.45	1
	0	7	.60	.75	1
	1	0-7	1.20	1.20	1
	2	0-7	1.20	1.20	1
	3	0-7	1.65	1.65	2
	4	0-7	1.35	1.35	1
	5	0-7	1.80	1.80	2
	6	0-7	1.50	1.65	2
	7	0-7	1.95	2.10	3

Single Operand

Instruction (Use with DST Time)	EF TIME (DST Mode = 0)	Memory Cycles	EF Time (DST Mode 1 to 7)	Read Memory Cycles
CLR, COM, INC, DEC, ADC, SBC, ROL, ASL, SWAB, SXT	.30 (J)	1	1.20	1
NEG	.75	1	1.50	1
TST	.30 (J)	1	.45	1
ROR, ASR	.30 (J)	1	1.20 (H)	1
ASH, ASHC	.75 (I)	1	.90 (I)	1

NOTE (H): Add 0.15 μsec if odd byte.
NOTE (I): Add 0.15 μsec per shift.
NOTE (J): Add 0.30 μsec if DST is R7.

Instruction (Use with SRC Times)	EF Time	Read Memory Cycles
MUL	3.30	1
DIV		
by zero	.90	1
shortest	7.05	1
longest	8.55	1

Instruction	EF Time	Read Memory Cycles	
MFPI	1.50	1	use with SRC times
MFPD	1.50	1	

Instruction	DST Mode	Instruction Time	Read Memory Cycles
MTPI MTPD	0	.90	1
	1	1.65	2
	2	1.65	2
	3	2.10	3
	4	1.80	2
	5	2.25	3
	6	2.10	3
	7	2.55	4

Branch Instructions

Instruction	Instr Time (Branch)	Instr Time (No Branch)	Read Memory Cycles
BR, BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BOS, BGE, BLT, BGT, BLE, BHI, BLOS, BHIS, BLO	.60	.30	1
SOB	.60	.75	1

Jump Instructions

Instruction	DST Mode	Instr Time	Read Memory Cycles
JMP	1	.90	1
	2	.90	1
	3	1.20	2
	4	.90	1
	5	1.35	2
	6	1.05	2
	7	1.50	3
JSR	1	1.95	1
	2	1.95	1
	3	2.25	2
	4	1.95	1
	5	2.40	2
	6	2.10	2
	7	2.55	3

Control, Trap & Miscellaneous Instructions

Instruction	Instr Time	Read Memory Cycles
RTS	1.05	2
MARK	.90	2
RTI, RTT	1.50	3
SET N, Z, V, C CLR, N, Z, V, C	.60	1
HALT	1.05	0
WAIT WAIT Loop for a BR is .3 μsec.	.45	0
RESET	10ms	1
IOT, EMT, TRAP, BRT	3.30	3
SPL	.60	1
INTERRUPT First Device	2.31	2

The following assembly language program was written for the PDP 11/70 computer to spatially filter or smooth data in an M X N array. The total execution time is equal to the sum of t_{INIT}, t_{a1}, and t_{C1} where:

t_{INIT} = 25.95 microseconds

t_{a1} = 1.65 - .9 microseconds

t_{b1} = 1.65 mn - 3.3n + 4.35 m - 8.7 microseconds

t_{C1} = 20.4 (m-2) (n-2) microseconds.

T_1 = 22.05 mn - 41.1 n - 34.8m + 72 microseconds

```
FILTER macro v1AUG75 18-nov-77 02:55 page 1

                                .title  filter
  000000 016567 filter::mov     2(r5),xaddr
         000002
         000106
  000006 016567         mov     4(r5),yaddr
         000004
         000102
  000014 017567         mov     @6(r5),m
         000006
         000066
  000022 017567         mov     @10(r5),n
         000010
         000062
  000030 017567         mov     @10(r5),incr
         000010
         000050
  000036 006367         asl     incr
         000044
  000042 066767         add     incr,xaddr
         000040
         000044
  000050 062767         add     #2,xaddr
         000002
         000036
  00056 066767          add     incr,yaddr
         000024
         000032
  00064 062767          add     #2,yaddr
         000002
         000024
  00072 012703          mov     #2,r3
         000002
  00076 012704          mov     #2,r4
         000002
  00102 000167          jmp     a1
         000014
  00106 000000 incr:    .word   0
  00110 000000 m:       .word   0
  00112 000000 n:       .word   0
  00114 000000 xaddr:   .word   0
  00116 000000 yaddr:   .word   0
  00120 000000 temp:    .word   0
  00122 026703 a1:      cmp     m,r3            ;last row?
         177762
  00126 001001          bne     b1              ;no
  00130 000207          rts     pc              ;return from subroutine
  00132 026704 b1:      cmp     n,r4            ;last col?
         177754
  00136 001013          bne     c1              ;no
  00140 012704          mov     #2,r4
         000002
  00144 005203          inc     r3              ;set address
  00146 062767          add     #4,xaddr        ;of pic1(i+1,2)
         000004
         177740
```

```
00154 062767          add     #4,yaddr          ;and pic2(i+1,2)
      000004
      177734
00162 000167          jmp     a1
      177734
                ;+
                ; spatial filtering on interior points
                ; of the picture matrix using 3x3
                ;-
00166 016702  c1:     mov     xaddr,r2
      177722
00172 016201          mov     -2(r2),r1         ;pic1(i-1,j)
      177776
00176 066201          add     2(r2),r1          ;+ pic1(i+1,j)
      000002
00202 166702          sub     incr,r2
      177700
00206 061201          add     @r2,r1            ;+ pic1(i,j-1)
00210 066201          add     -2(r2),r1         ;+ pic1(i-1,j-1)
      177776
00214 066201          add     2(r2),r1          ;+ pic1(i+1,j-1)
      000002
00220 016702          mov     xaddr,r2
      177670
00224 066702          add     incr,r2
      177656
00230 061201          add     @r2,r1            ;+ pic1(i,j+1)
00232 066201          add     -2(r2),r1         ;+ pic1(i-1,j+1)
      177776
00236 066201          add     2(r2),r1          ;+ pic1(i+1,j+1)
      000002
00242 006201          asr     r1
00244 006201          asr     r1
00246 006201          asr     r1
00250 010177          mov     r1,@yaddr
      177642
00254 062767          add     #2,yaddr
      000002
      177634
00262 062767          add     #2,xaddr
      000002
      177624
00270 005204          inc     r4
00272 000167          jmp     b1
      177634
      000000'         .end    filter
```

The execution times of instructions, for the microcomputer system built using the AM 2900 series bit slice family, are shown below. These instruction times assume a data and program memory cycle of 375 nanoseconds.

Microcomputer Instruction Set

Register-to-Register Instructions	
Add	
Subtract	
Negate	
Transfer	0.55 μs
Clear	
Increment	
Decrement	
Complement	
Other Instructions	
Logical Shift	0.85 μs + 0.15 μs Per Bit
Arithmetic Shift	
Skip on Flags	0.70 μs
Input	1.3 μs
Output	
No Operation	0.55 μs
Memory Reference Instructions	
Load	
Store	0.95 μs
Add	
Subtract	
Multiply	3.55 μs
Divide	4.30 μs
Compare	0.95 μs
Limit	1.3 μs
Memory Increment	1.1 μs
Memory Decrement	
AND	
OR	0.95 μs
Exclusive OR	
Branch Unconditionally	0.55 μs
Branch to Subroutine	0.70 μs

The following program, written in HP assembler language is for smoothing an M X N array. The HP instructions correspond very closely to the microcomputer instruction set. Therefore, a translation was made between the instruction sets. Two assumptions were made. The first assumption is that each level of indirect addressing adds .5 microseconds. Also, the HS "ISZ" instruction corresponds to the combination of the following three microcomputer instructions: "Increment," "Compare," and "Skip."

The program was analyzed and the following timing equation was obtained.

$$T_2 = 30.2 \text{ mn} - 60.4 \text{ (m-n)} + 100 \text{ microseconds.}$$

```
ASMB
*       THIS IS A PROGRAM IN HP ASSEMBLER LANGUAGE FOR SMOOTHING A
*       PICTURE DIGITIZED IN AN N*N MATRIX.
*
*       IT IS ASSUMED THAT THE ELEMENTS OF THE INPUT DIGITAL PICTURE
*       ARE STORED IN A VECTOR FROM TOP-LEFT TO BOTTOM-RIGHT.
*       COLUMN BY COLUMN (E.G. THE ELEMENT A(1.1) OCCUPIES THE
*       ADDRESS 1.THE ELEMENT A(1.2) THE ADDRESS N+1 . THE ELEMENT
*       A(2.1) THE ADDRESS 2 ETC.)
*
        NAM PEND.7
        ENT PEND
        EXT .ENTR            INSTRUCTIONS NEEDED FOR TRANSFERRING PARAMETERS
C       NOP                 FROM FORTRAN MAIN PROGRAM TO ASSEMBLER
D       NOP                  SUBROUTINE (TWO MATRICES ARE TRANSFERRED : C AND D)
PEND    NOP
        JSB .ENTR
        DEF C
*
        LDB D               INSTRUCTIONS FOR PREPARING REGISTERS.
        ADB NP1             THEY ARE OUT OF LOOPS.
        STB DI              SOME INSTRUCTIONS HERE AND ALONG THE
        LDB KI               PROGRAM ARE NEEDED BECAUSE SMOOTHING IS NOT
        STB K                APPLIED TO THE 4*(N-1) BORDER ELEMENTS
        LDB C               OF THE INPUT MATRIX.
        ADB NP1             THE ELEMENTS TO WICH THE ALGORITHM IS
        STB CI               APPLIED WILL BE CALLED "ELEMENTS OF INTEREST"
*
*
LD2     ADB NM1             GO TO THE ADDRESS OF THE ELEMENT PLACED
*                           AT NORTH-EAST OF THE CONSIDERED ONE.
        LDA 1.I              STORE ITS VALUE.
        INB                 GO TO THE EAST ELEMENT.
        ADA 1.I              ADD THE VALUES OF THE EAST AND NORTH-EAST ELEMENTS
        INB
        ADA 1.I
        ADB MN
        ADA 1.I
        ADB MN              SCAN IN SUCCESSION THE REMAINING 8-NEIGH-BOURS OF
        ADA 1.I             THE CONSIDEREL ELEMENT AND ADD THEIR VALUES
        ADB M1              TOGHETHER.
        ADA 1.I
        ADB M1
        ADA 1.I
        ADB N
        ADA 1.I
*
*
        ARS                 SHIFT 3 TIMES TOWARDS RIGHT. THE BINARY VALUE OF
        ARS                 THE SUM JUST OBTAINED. E.G.DIVIDE BY 8.
        ARS
*
*
        STA DI.I            RELABEL THE CONSIDERED ELEMENT.
*
*
        ISZ K               INCREMENT K AND IF K=0 SKIP NEXT INSTRUCTION AND
        JMP LD1             RESET K TO K1 OTHERWISE JUMP TO LD1.
        LDB K1               THESE INSTRUCTION ARE REQUIRED TO SKIP THE
        STB K                2*(N-2) ELEMENTS BELONGING TO THE FIRST AND
*                            LAST ROW OF THE INPUT MATRIX.
*
*
*
*                            WHEN K=0 THE "ELEMENTS OF INTEREST" OF ONE COLUMN
*                            OF THE INPUT MATRIX (I. E. THE VECTOR'S ELEMENTS
        LDB D1              FROM THE 2.ND TO THE (N-1).TH OR THE ONES FROM THE
        ADB D3              (2N+2).TH TO THE (3N-1).TH ETC.) HAVE ALL BEEN
        STB DI              CONSIDERED. TO INITIATE THE SCANNING OF THE NEXT
        LDB CI              COLUMN. THE CURRENT ELEMENT ADDRESS HAS TO BE
        ADB D3              INCREMENTED BY 3. A TEST IS ALSO PERFORMED TO
        STB CI              ASCERTAIN WHETHER SUCH AN INCREMENTATION HAS BEEN
        ISZ H                MADE (N-2) TIMES (IN THIS CASE THE MATRIX HAS BEEN
        JMP LD2              COMPLETELY SCANNED) OR NOT. ACCORDING TO THE
        JMP PEND.I           RESULT OF THE TEST. THE PROGRAM JUMPS TO THE
*                            SUBROUTINE PEND WHICH RETURNS TO THE FORTRAN
*                            MAIN PROGRAM "PRINT AND END". OR GOES TO LD2.
*
*
LD1     ISZ DI
        LDB CI
        INB                 GO TO THE NEXT ELEMENT OF THE VECTOR AND
        STB CI              APPLY TO IT THE ROUTINE LD2
        JMP LD2
*
*
NP1     DEC 11               INSTRUCTIONS DEFINING THE QUANTITIES USED IN THE
NM1     DEC 9               PROGRAM. THEY ARE OUT OF THE LOOPS.
M1      DEC -1              THE NUMERICAL VALUES GIVEN ON THE LEFT ARE
D3      DEC 3               DEDUCEDFROM THE FOLLOWING FORMULAS. FOR N=10.
K1      DEC -8
K       DEC 0               NP1=N+1         NM1=N-1         M1=-1
H       DEC -8              D3=3            K1=-(N-2)       K=0
N       DEC 10              H=-(N-2)        N=N
MN      DEC -10
DI      NOP
CI      NOP
        END
        ENDS
**** LIST END ****
```

BEST AVAILABLE COPY

References

1. Lillestrand, R. L. and R. R. Hoyt, "The Design of Advanced Image Processing Systems," Photogrammetric Engineering, 7, 16: 1201-1217, 1974.

2. H. S. Stone, Editor, Introduction to Computer Architecture, Science Research Associates, Chicago, Illinois (1975).

3. Mori, K., Shinoda, H. and Asada, H., Toshiba Pattern Information Cognition System - TOSPICS," Toshiba Review, No. 107, January-February 1977.

4. Cordella, L., M. J. B. Duff and S. Levialdi, "Comparing Sequential and Parallel Processing of Picture," Proc. IEEE Internation Conference on Pattern Recognition, 1976.

5. Ramsey, D. M. (Ed.) Image Processing in Biological Science. University of California Press, Berkeley and Los Angeles, California, 1968,

6. McCormick, B. H. , "The Illinois Pattern Recognition Computer - Illiac III," IEEE Trans. Electron. Comput., EC-12, No. 5, 1963, pp. 791-813.

7. Duff, M. J. B., "Cellular Logic and its Significance in Pattern Recognition, AGARD Conf. Proc. No. 94 on Artificial Intelligence, 25-1 (1971).

8. Duff, M. J. B., D. M. Watson, T. J. Fountain and G. K. Shaw, "A Cellular Logic Array for Image Processing," Pattern Recognition, Vol. 5, 1973, pp. 229-247.

9. Duff, M. J. B., D. M. Watson, "CLIP 3 Operating Manual, Internal Report 73/4," Image Processing Group, University College London.

10. Duff, M. J. B., D. M. Watson and E. S. Deutsch, "A Parallel Computer for Array Processing," Information Processing 74, Proc. IFIP Cong. 74, 94 (1974).

11. Duff, M. J. B., "CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor," Proc. IEEE Internation Conference on Pattern Recognition, 1976.

12. Duff, M. J. B., "CLIP 4: Internal Report," Image Processing Group, University College London.

13. Cordella, L., Duff, M. J. B., and Levialdi, S., "Thresholding: A Challenge for Parallel Processing," Computer Graphics and Image Processing 6, 207-220 (1977).

14. Weimann C. F. R., and Grosch, C. E., "Parallel Processing Research in Computer Science: Relevance to the Design of a Navier-Stokes Computer" 1977 International Conference on Parallel Processing.

15. Gritton, E. C., et. al., Feasibility of a Special-Purpose Computer to Solve the Navier-Stokes Equations," RAND Corporation, R-2183-RC, (June 1976), 74 pp.

16. Dew. B. et. al., "An Automatic Microscope System for Differential Lenkocyte Counting," Journal of Histochemistry and Cytochemistry, Vol. 22, No. 7, pp. 685-696.

17. Economidis, T., "The Illiac IV System," NASA/AMES Research Center.

18. H. J. Siegel, Interconnection Networks and Masking Schemes for Single Instruction Stream - Multiple Data Stream Machines, Dept. of Electrical Engineering and Computer Science, Princeton University, Ph.D. Thesis (May, 1977).

19. T. Feng, "Data Manipulating Functions in Parallel Processors and their Implementations," IEEE Trans. Comput., Vol. C-23 (March 1975), pp. 309-318.

20. W. J. Bouknight, et. al., "The Illiac IV System," Proceedings of the IEEE, Vol. 60, No. 4 (April 1972), pp. 369-388.

21. K. E. Batcher, "The Flip Network in STARAN," Proceedings of the 1976 International Conference on Parallel Processing (August 1976), pp. 65-71.

22. Pease, M., "An Adaptation of the Fast Fourier Transform for Parallel Processing," Journal of ACM, Vol. 15, No. 2, April 1968, pp. 252-264.

23. H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Comput., Vol. C-20 (February 1971), pp. 153-161.

24. Lowe, E., "A 16-bit Microcomputer for Missile Guidance and Control Applications," Proc. of JACC 1977.

25. The AM 2900 Family Data Book, Advanced Micro Devices, Inc., Sunnyvale, California 94086.

26. Intel Data Catalog 1975, Intel Corporation, Santa Clara, California 95051.

27. "4 Bit Expandable Bipolar Microcontroller 5701/6701" Monolithic Memories Inc., Sunnyvale, California 94086.

28. Ghest, C.," A Powerful Microprogram Control Unit-6700," Monolithic Memories Inc., Sunnyvale, California 94086.

29. Bipolar Microcomputer Components Data Book, Texas Instruments Inc., Dallas, Texas 75222

30. Blood, W., "M10800-The High Performance LSI Processor Family," Motorola, Inc., Phoenix, Arizona 85036.

31. McWilliams, T. M., "Using LSI Processor Bit-Slices to Build a PDP-11 - A Case Study in Microcomputer Design," National Computer Conference, 1977.

32. A Pocket Guide to Hewlett-Packard Computers, Hewlett-Packard, Palo Alto, California 94304.

33. Akers, A. E., "An Auxiliary Memory Controller for Array Processing," TR-EE 76-3, School of Electrical Engineering, Purdue University.

34. H. J. Siegel, "Controlling the Active, Inactive Status of SIMD Machine Processors," 1977 International Conference on Parallel Processing. August 1977

35. Organick, E. I. Computer System Organization: THE B5700/B6700 Series, ACM Monograph Series.

36. PDP 11/70 Processor Handbook, Digital Equipment Corporation, Maynard, Massachusetts 01754.

BIBLIOGRAPHIC DATA SHEET	1. Report No. TR EE 77-42	2.	3. Recipient's Accession No.
4. Title and Subtitle Preliminary Design of a Partitionable Multi- Microprogrammable Microprocessor System for Image Processing			5. Report Date 11/21/77 6.
7. Author(s) Julius Bogdanowicz			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Purdue University School of Electrical Engineering West Lafayette, IN 47907			10. Project/Task/Work Unit No. 11. Contract/Grant No. AFOSR 74-2661 NSF ENG 76-18567
12. Sponsoring Organization Name and Address AFOSR Bldg. 410 Bolling AFB Washington, D.C. 20332	National Science Foundation 1800 G. Street, NW Washington, D.C. 20550		13. Type of Report & Period Covered 14.
15. Supplementary Notes			

16. Abstracts

An approach to image processing hardware based on the concept of cellular logic arrays is first evaluated. A preliminary design of a partitionable multi-microprogrammable microprocessor system for image processing is then described. The performance of the multi-microprogrammable microprocessor system is also investigated.

17. Key Words and Document Analysis. 17o. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 68
	20. Security Class (This Page UNCLASSIFIED	22. Price

-69